Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institute of Scientific Computing
Computer Graphics Group

CS Technical Report #335, February 28, 2000

A Survey of Surface Representations for Geometric Modeling

Andreas Hubeli, Markus Gross

Computer Science Department
ETH Zurich, Switzerland
e-mail: {hubeli, grossm}@inf.ethz.ch
http://graphics.ethz.ch

A Survey of Surface Representations for Geometric Modeling

Andreas Hubeli **Markus Gross**
ETH Zurich
Switzerland

Abstract

In this survey we will describe the most important state-of-the art surface representations, and we will investigate their usefulness as modeling tools in geologic applications. The representations that we will analyze include wavelet based representations, hierarchical splines, subdivision surfaces, mesh based methods, schemes based on signal processing tools and classic representations used in geoscience. These representations will be evaluated using the most important requirements that a modeling system in geoscience should fulfill. We conclude the survey with the description of some interesting research topics to improve on the representations currently available.

1 Introduction

The representation of complex surfaces has been one of the core fields of computer graphics, and over the years many different approaches to this problem have been constructed.

The word *representation* should not be identified with the data structures that allow to render complex meshes very fast and very accurately, since this is just one of the requirements that a good representation must fulfill. Other requirements include being able to compute intersection curves between surfaces, model the error, edit surfaces at different levels of resolution, fitting surfaces through clouds of points, and many more. A complete list of the most important requirements for a surface representation in geoscience applications can be found in Section 2.
None of the representations currently available is capable to meet all the surface requirements that an application might specify. Some of the requirements are not compatible: constructing compact representations is often very important, but storing some redundancy in the data usually allows to construct faster algorithm. As a consequence the current representations concentrate on meeting a few important requirements.
In this survey we are going to analyze the state-of-the-art surface representations currently available as well as the most influential classic representations. This analysis includes a description of the theory that defines the representations, the operators used in the algorithms, and an evaluation based on a set of requirements.

Figure 1.1 gives a taxonomy of the representations investigated in this survey. They can be grouped into six categories:

1. *Wavelet based representations*: Wavelets are very well understood from a mathematical point of view, and they are defined in a rigorous framework.
   This category includes standard first generation tensor product wavelets, second generation wavelets, the BLaC wavelets and wavelets over manifolds.
2. *Hierarchical splines*: The H-Spline representation is a pioneer work in hierarchical representation of surfaces. This representation is based on the B-Spline basis function.
3. *Subdivision surfaces*: Subdivision operators generate smooth surfaces from polygonal meshes, which by definition are piecewise linear. The first papers on subdivision are more than twenty years old, but new papers are still improving the representation.

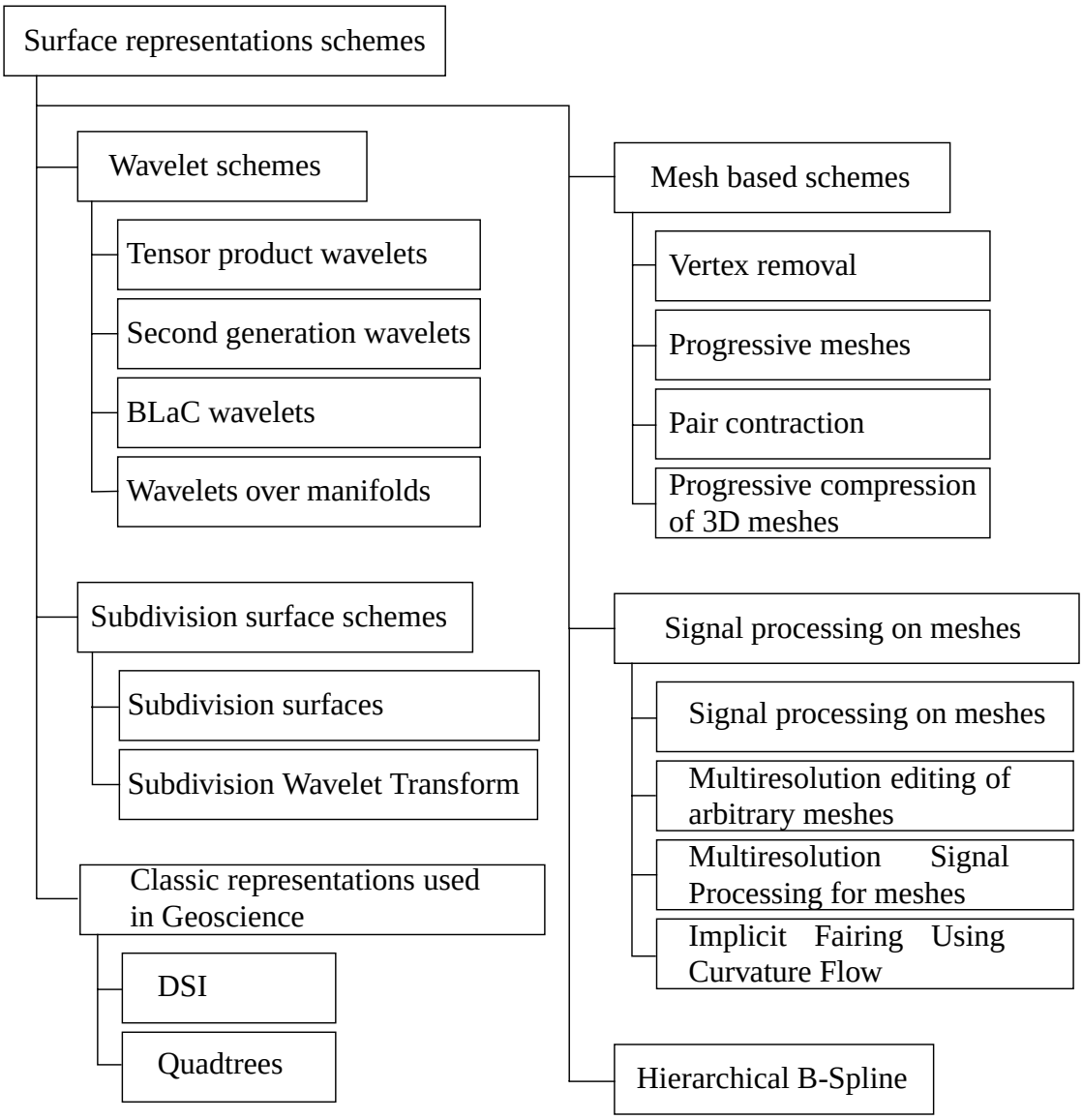


Figure 1.1 A taxonomy of representations analyzed in this survey

4. *Mesh based methods*: All the representations that work on piecewise linear representation of surfaces, which are either triangular or polygonal meshes, are listed under this category. Wavelets, H-Splines, subdivision schemes and also signal processing methods are not mesh based methods since they represent surfaces using higher order functions. Some of the better known algorithms such as progressive meshes and the classic vertex removal algorithm belong to this category.
5. *Signal processing schemes*: This family of representations apply discrete operators based on signal processing tools on meshes with arbitrary connectivity.
   These representations construct fairing operators to smooth meshes, and they use classic mesh based approaches to store the meshes.
6. *Classic representations used in geoscience*: there are two main representations used in geoscience. The DSI algorithm allows to create surfaces that satisfy a number of user-specified constraints, such as positional and higher order constraints on the vertices of a mesh. Since the most commonly used data format in geoscience is the regular grid, quadtree structures are often used to construct efficient representations.

In Section 2 we will list the most important requirements that a representation must fulfill to optimally represent geologic models. The different requirements will be described and weighted depending on their importance. In Section 4 through Section 9 we will describe the different representations and evaluate them using the requirements listed in Section 2. In Section 10 we will give some conclusions and directions for future work.

2 Surface Representation Requirements

The development of complex surface representation schemes has been one of the core fields of computer graphics and geometric modeling. The different representations currently available have succeeded in modeling certain properties of surfaces well, but none of them was general enough to satisfy all the requirements that could be imposed on a representation.

Since no representation is able to satisfy every modeling and visualization requirement, we need to define a set of crucial surface requirements, and weight them with respect to their importance. This step will allow us to evaluate all surface representations and to rank them with respect to their usefulness in representing geologic models.

Table 1 summarizes the most useful requirements needed to represent geologic models:

Requirements	**Importance**
Modeling of two-manifold surfaces with boundaries	1
Computation of surface-surface intersections	1
Scalable representation	1
Modeling of non-manifold singularities, tears and cracks	2
Error modeling	2
Smoothness of the surface	3
Multiresolution editing	3
Surface fitting	3
Support of local high variation in the curvature of the surface	4
Scale independence	4
Changes of the surface over time	4

TABLE 1. Requirements for the surface representations

The smaller the importance value associated with each requirement, the more important the requirement is: requirements with an importance of one are mandatory, while requirements with an importance of four describe less crucial features.

In the remaining of this section the requirements listed in Table 1 will be described in detail.

2.1 Modeling of Two-Manifold Surfaces With Boundaries

In order to properly describe what a two-manifold is, the concepts of macrotopology and microtopology need to be defined.

- *Macrotopology*: describes the set of topologic properties that are independent of the discrete representation of surfaces. The definition of a two-manifold and the genus of a surface are two examples of macrotopology.

- *Microtopology*: describes the set of topologic properties that depend on the discrete representation of surfaces. The definition of a pseudo-manifold and the concept of a simplex connectivity are two examples of microtopology.

Surfaces can be classified according to their complexity: the simplest surfaces are defined as *height field over regular grids*, the most complex surfaces are *non-manifolds surfaces*. One important question that has to be answered is what type of surfaces must be supported by the representation.

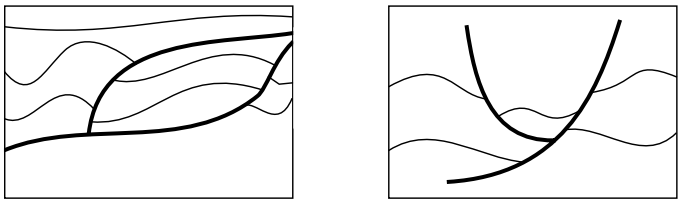
In order to be able to build geologic models we have to be able to model at least *two-manifold surfaces with boundaries*. A two-manifold can be interpreted intuitively as a surface that does not intersect itself. A more rigorous definition is given in Definition 2; additional information can be found in [63]:

Definition 1: *A homeomorphism is defined as a continuous invertible map* $f: K \to H$ *whose inverse* $f^{-1}: H \to K$ *is also continuous.*

Definition 2: *An n-dimensional manifold with boundary* M *is a Hausdorff space such that each point has an open neighborhood homeomorphic to* R^n *or to* $R^n_+ = \{(x_1, \ldots, x_n) \in R^n \,|\, x_n \geq 0\}$. *A two-manifold with boundary is an n-dimensional manifold with boundary with* $n = 2$.

Two-manifold surfaces are powerful enough to model simple horizons and faults that are usually represented with regular grids as well as more complex objects such as salt domes. Furthermore a representation that supports two-manifold surfaces with boundaries is capable of storing most of the surfaces that are generated by the intersection of two surfaces.

Two-manifold surfaces are not capable to represent some of the most complex faults and horizons, such as the examples shown in Figure 2.1:



a) b)

Figure 2.1 Complex non-manifold geologic surfaces
a) Thrust duplex structure
b) Normal and inverse faults

Figure 2.2 illustrates a real world example of a complex geologic surface, a folded shale. The picture was taken in Brienz, Switzerland.

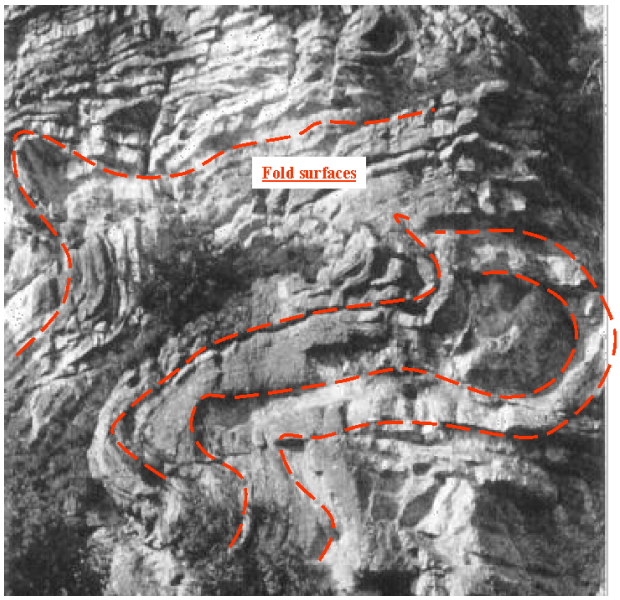


Figure 2.2 Folded shale (picture courtesy of H. Einstein and V. Ivanova)

If this type of non-manifold surfaces are present in a geologic model, it is possible to decompose them in two-manifold surfaces, for example using the algorithm developed by Gueziec et al. in [1].

2.2 Computation of Surface-Surface Intersections

A geologic model is built by first selecting a volume of interest and then inserting geologic objects such as faults and horizons into the volume. This process is shown in Figure 2.3; for a more accurate description of the problem of building geologic models refer to [70] and [72].

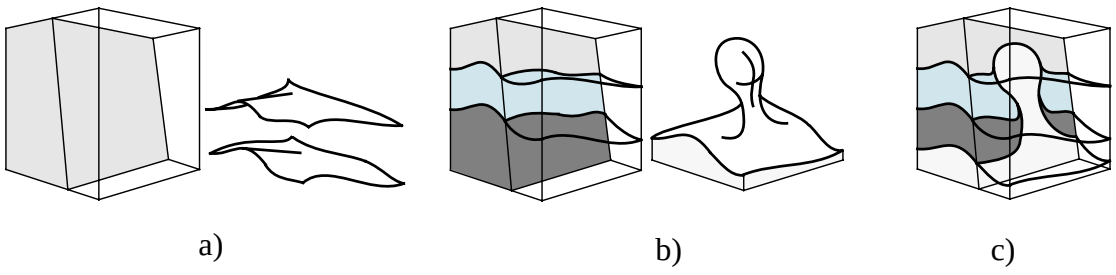


Figure 2.3 Construction of a geologic model
a) Subdivide a region of interest with surfaces
b) Insert a subvolume into the result
c) Mark features such as layers, horizons, and salt bodies

Figure 2.4 illustrates how a geologic model is built using the Common Model Builder (CMB), a modeling framework developed by Schlumberger. Additional information on the CMB framework can be found in [58].

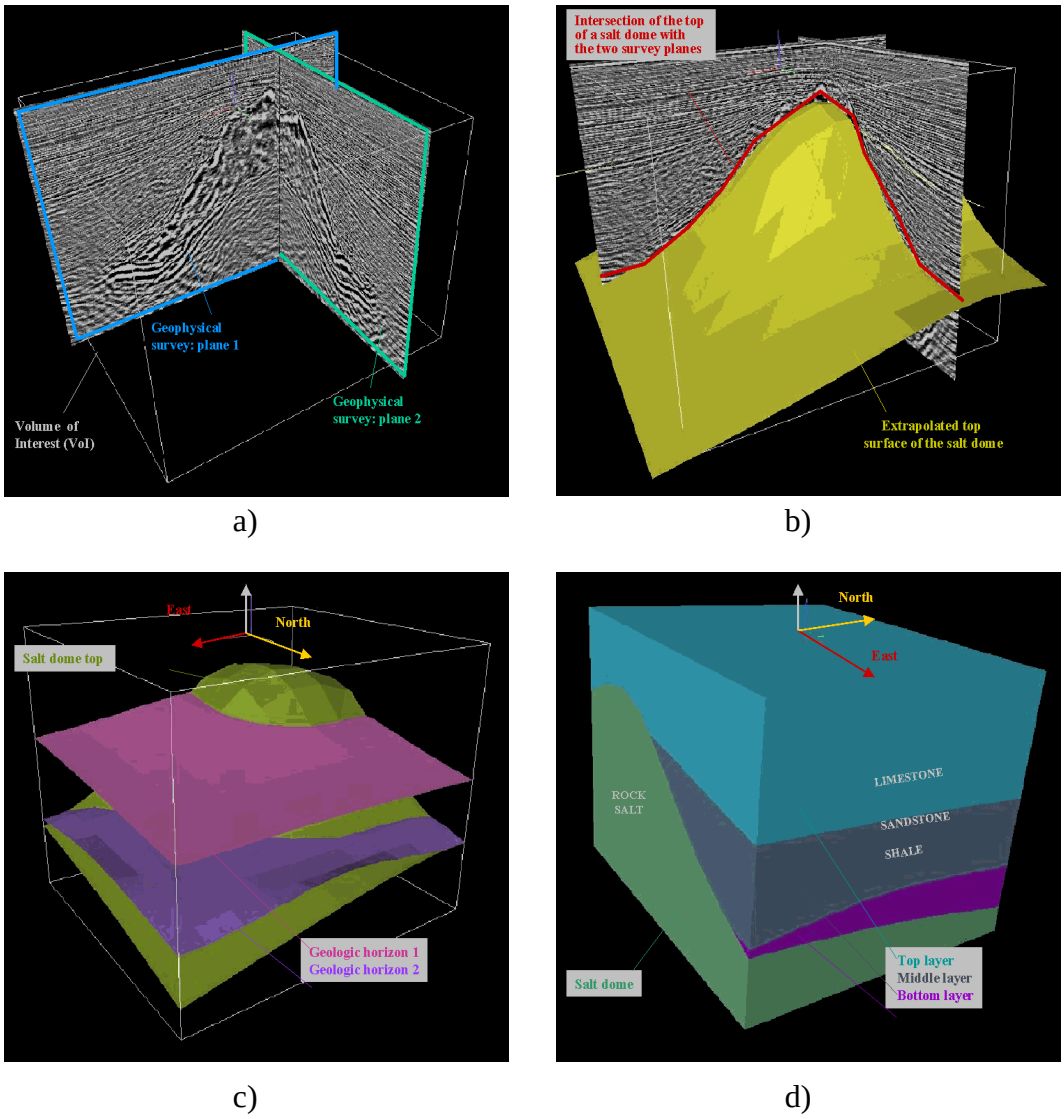


Figure 2.4 **A model in CMB (pictures courtesy of Schlumberger)**
a) Seismic data extracted from a geologic survey
b) A surface is fitted through the seismic data
c) The model after three surfaces have been introduced
d) The surfaces in the model bounds regions with different material properties

The information on these models is stored in a boundary representation. This means that a 3D volume is defined by its 2D boundaries, 2D surfaces are defined by their 1D bound-

aries and 1D curves are defined by their 0D boundaries. An example of a boundary representation is described in Figure 2.5.

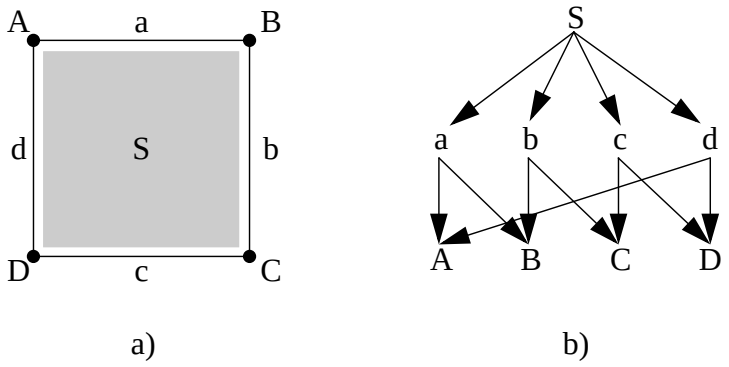


Figure 2.5 Boundary representation of a simple model
a) Labeled model
b) Boundary representation of the model

The information needed to keep a consistent boundary representation of geologic models is computed via surface-surface intersection algorithms. These algorithms are used when a fault or horizon is added in the model, or when one of the surfaces already in the model is edited.

2.3 Scalable Representation

The complexity of geologic models varies greatly: some models are relatively simple and are described with a few thousands primitives, other models are much more complex and are described with millions of primitives.
A good surface representation must be able to handle surfaces of any complexity, i.e. it should fulfill the following requirements:

- The time required to build a representation of a surface must have a low complexity, equal or smaller than $O(n \cdot \log n)$, where n corresponds to the number of vertices in the piecewise linear approximation of the surface.
- The storage requirements for a representation of a surface must also have a low complexity, equal or smaller than $O(n \cdot \log n)$, where n corresponds to the number of triangles in the piecewise linear approximation of the surface.
- Once the representation of the surface has been constructed and stored it must be possible to generate error-bounded approximations of the surface. The quality of the approximations must depend on two factors:
    1. It must be possible to interact with the model. Since it is currently not feasible to interact with full resolution models, the representation has to return an approximation of the model that minimizes the error norm, but still allows interaction with the model.

2. The representation must scale with the available hardware: the approximations on less powerful workstations will be coarser than the representation on powerful supercomputers.

The importance of scalable surface representations is growing steadily, since data acquisition systems are able to extract more and more information and build better and more complex models.

2.4 Modeling of Non-Manifold Singularities, Tears and Cracks

A representation that can model two-manifolds is capable of modeling most of the surfaces commonly found in a geologic model. However other primitives exist that would be useful to model:

- As we have seen in Section 2.1 a representation that can model two-manifold surfaces is powerful enough to model most geologic models. There is one particular instance of non-manifold surfaces that we would like to model: pseudo-manifold singularities, which are defined in Definition 3 as well as in [63].

Definition 3: *An n-dimensional pseudo-manifold is an n-dimensional finite, regular CW-complex which satisfies the following three conditions:*

1. *Every face is a face of some n-cell*
2. *Every* $(n-1)$*-dimensional cell is a face of exactly two n-cells*
3. *Given any two n-cells* e *and* e' *there exist a sequence of n-cells* $e_0, \ldots, e_k$ *such that* $e_0 = e$, $e_k = e'$ *and each pair* (e_{i-1}, e_i) *has a common* $(n-1)$*-dimensional face.*

The vertex v in Figure 2.6 is an example of a pseudo-manifold singularity:

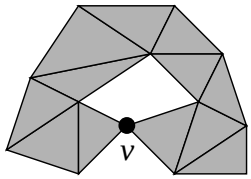


Figure 2.6 Pseudo-manifold surface

The surface drawn in Figure 2.6 is not a two-manifold, since the vertex v does not have a neighborhood homeomorphic to either R^2 or R^2_+, but it is a pseudo-manifold, since it satisfies the conditions of Definition 3.
These vertices are obviously very important: the removal of v from the mesh would not only change the connectivity locally, but it would also change it globally, since the hole in the surface would no longer exist.

- The second type of objects we would like to represent includes tears and cracks. These objects could be represented easily if we could embed curves in the surfaces. Embedded curves could also be used to represent intersections.

2.5 Error Modeling

Error modeling is an essential component of a surface representation that gives the user feedback on the quality of the approximated model he is interacting with.

Real-time interaction with full-resolution geologic models is often not feasible, since the models are too complex and they are represented with too many primitives; it is often necessary to build simpler approximations that can be handled in real-time. Specific information on the requirements of a scalable representation can be found in Section 2.3.

The simpler models do not contain all the information of the full-resolution models. The differences between the approximations and the original models are introduced by:

- reducing the number of primitives, usually triangles, in the model
- approximating the location of the vertices in the model

The goal of an error modeling system is to give the user an insightful application-dependent feedback on the quality of the approximations.
As an example, a geologist visualizing a model might be interested in an error norm that describes the difference between the approximated and the original full-resolution model from a visualization point of view. On the other hand during a flow simulation the error norm could describe the differences between the volumes defined in the approximated model and the volumes in the full resolution model.

2.6 Smoothness of the Surface

The mesh that describes a surface can be interpreted in two different ways:

- the mesh fully defines the surface. The surface is identified with the collection of polygons and vertices that specifies the mesh.
  Representations based on this assumption can usually model surfaces with any connectivity, but they are not very well suited to model the error and to edit surfaces.
- the mesh is used as a constraint to build the surface representation, which is defined as a piecewise defined smooth function.
  Representations based on this assumption are usually restricted to simple connectivity, such as quaternary subdivision meshes, but they are better suited to model the error and to edit surfaces.

Since there is no unique surface that interpolates a given set of points, there is some freedom in the choice of the surface. We choose to construct smooth surfaces, with non-smooth regions only where needed. Among other benefits, this choice allows to easily define which component in the input data is noise.

2.7 Multiresolution Editing

Editing is an integral component of a surface representation for geoscience. The most important application of a multi-resolution editing tool is the ability to enforce geologic constraints to models that were generated automatically using a surface fitting algorithm. This is a crucial step in the construction of a geologic model, since fitting algorithms build geometrically consistent models, but they cannot guarantee that geologic constraints are met.

Multiresolution editing tools make the editing easier by allowing the user to edit surfaces at different levels of resolution, which means that the support of the edit operations can be made larger or smaller depending on the type of edit needed.
Furthermore multiresolution editing tools edit either smoothed polygonal meshes or higher order functions, so that a surface generated by an edit operation is as smooth as the surface before the edit, and an edit affects the correct frequency interval of a surface.

2.8 Surface Fitting

There are two principal types of data used to construct geologic models:

- *Borehole data*: the data is measured in a borehole. This type of data is precise and dense in depth, but it is very sparse, since the distance between boreholes is large.
- *Seismic data*: the data is acquired by emitting sound waves into the ground and analyzing their reflections. These datasets are much more dense, but less precise.

The borehole and seismic information is not directly available as a set of faults and horizons that define the geologic model in the boundary representation; instead the information is specified with sets of points or sets of curves. These points and curves must then be connected together to form surfaces, and this is the goal of the surface fitting methods.

2.9 Support of Local High Variation in the Curvature of the Surface

As we have seen in Section 2.6 one of the surface requirements is to generate smooth surfaces using (piecewise) smooth functions. We have chosen to construct smooth surfaces, since the resulting models will look more natural. While surfaces should be represented as smooth objects, portions of the surfaces might not be smooth. These non-smooth regions should be represented correctly by the surface representation.
Ridges and horizons cut by faults are two examples of non-smooth regions.

2.10 Scale Independence

Geologic data ranges from millimeters to kilometers: as already discussed in Section 2.8 the distance between boreholes is usually measured in kilometers, but the data that is collected in a borehole has a resolution of a few millimeters. A surface representation must therefore be independent of any scale and be able to represent very high resolution data as well as coarse data.

All the representations that are going to be analyzed in this survey fulfill this requirement, and we will not discuss it further.

2.11 Changes of the Surface Over Time

The geologic models are built from the information generated from the acquisition systems. These models describe portions of the earth crust at a given point in time. This information is then used to describe flows of gases and liquids, such as oil, gas and water. In order to model this information it is necessary to extend a surface representation to four dimensions: three dimensions describe the geometric position of the vertices, one dimension specifies the temporal position of the model.

3 Introduction to the Surface Representations Schemes

In the following six sections we will introduce the most important state-of-the-art surface representations available today. We will describe these representations and evaluate them with respect to the requirements introduced in Section 2; more details on the representations can be found in the original papers.

Since there are so many representations currently available, it is not feasible to present them all separately, therefore we grouped them into different categories. For each of these categories the basic mathematical background will be described, as well as some of the most interesting results. Due to the sheer number of papers and theories in this field we do not claim completeness, unintentionally some representations may have been left out.

The representations that will be discussed in this survey can be categorized in many different ways. For our purposes we decided to distinguish between representations that work on:

- *Higher order functions*: the surfaces are not defined by the polygonal meshes, but by piecewise defined higher order functions. The subdivision surface schemes are one well know example: although the basic components of this representation are polygons, in the limit these schemes generate provably smooth surfaces.
- *Polygonal meshes*: these algorithms work directly on the mesh and build data structures to efficiently store, access and visualize meshes.

  Note that the polygonal meshes can also be interpreted as linear C^0 approximations of smooth surfaces.

The two approaches have been mutually exclusive up until recently: a representation either worked on a polygonal mesh or it work on higher order functions. This is not true anymore: it is important to notice that some of the most recent works are trying to unify these two different approaches by using signal processing tools on meshes with arbitrary connectivity.

4 Wavelet Based Representations

4.1 Tensor Product Wavelets

In this section we give an overview of the classical multiresolution analysis applied on functions defined over R^n in the context of wavelet transform. We first describe the basics of multiresolution analysis, and then give some examples of basis functions that could be used to represent surfaces in geoscience.

The theory presented in this section is meant to give an overview of the vast field of multiresolution analysis and wavelets. More details on multiresolution analysis, wavelets and filter banks can be found in the original paper of Mallat [61], in the classic books of Chui [10], Daubechies [16], Yves Meyer [64], Eric Stollnitz et al. [78] and [79], as well as in [65], [80], [12], [4] and [60].
A surface representation framework based on first generation wavelet is described in [39].

The results presented in this section allow to construct a multiresolution representation of surfaces defined as height fields over regular grids. As a consequence it is possible to model most faults and horizons present in a geological model, since most of these surfaces are currently represented using this regular grids.

4.1.1 Definitions

The classic wavelet theory analyzes the space $L^2(R)$ of all *measurable functions* $f(x)$ defined in Eq. 4.1:

$$\int_{-\infty}^{\infty} |f(x)|^2 dx < \infty \qquad \text{(EQ 4.1)}$$

In a similar way Eq. 4.2 defines the space $l^2(R)$ of all *bi-infinite square-summable sequences* $\{c_i, i \in Z\}$:

$$\sum_{i=-\infty}^{\infty} |c_i|^2 < \infty \qquad \text{(EQ 4.2)}$$

One integral component of a multiresolution analysis is the *inner product* between two functions $f(x), g(x) \in L^2(R)$ defined as

$$\langle f(x), g(x) \rangle = \int_{-\infty}^{\infty} f(x) \cdot \overline{g(x)} dx \qquad \text{(EQ 4.3)}$$

Eq. 4.3 can be used to define the *two-norm* of a function $f(x)$:

$$\|f(x)\|_2 = \langle f, f \rangle^{1/2} \quad \text{(EQ 4.4)}$$

Furthermore, the *Kronecker function* is defined as

$$\delta_{j,k} = \begin{cases} 1 \text{ for } j = k \\ 0 \text{ for } j \neq k \end{cases} \quad \text{(EQ 4.5)}$$

A sequence of elements $\{e_i, i \in Z\}$ in an *Hilbert space* H forms a *Riesz basis* of H if:

1. For any sequence of elements $\{c_i, i \in Z\} \in l^2(R)$ there exists constants $\infty > b_2 \geq b_1 > 0$ such that

$$b_1 \cdot \|\{c_i\}\|_2^2 \leq \left\| \sum_i c_i \cdot e_i \right\|_2^2 \leq b_2 \cdot \|\{c_i\}\|_2^2 \quad \text{(EQ 4.6)}$$

2. The vector space $\sum_i c_i \cdot e_i$ is dense in H.

For a definition of the Hilbert spaces see [71] and [76].

4.1.2 Multiresolution Analysis

We start our investigation by constructing a function $\psi(x)$ that can generate $L^2(R)$.

The first property of $\psi(x)$ can be extracted directly from Eq. 4.1: the value any function $f(x) \in L^2(R)$ must either decay exponentially to zero towards $\pm\infty$ or have a compact support.

The most straightforward strategy that allows $\psi(x)$ to cover R is to consider its *integral shift*:

$$\psi(x-k), k \in Z \quad \text{(EQ 4.7)}$$

The function $\psi(x)$ must also be able to capture the information of any function $f(x) \in L^2(R)$ at different levels of detail, from small scale changes in the function to large scale changes. This information can be captured by *binary dilation* of $\psi(x)$:

$$\psi(2^j \cdot x), j \in Z \quad \text{(EQ 4.8)}$$

From Eq. 4.7 and Eq. 4.8 a new set of functions can be defined:

$$\psi_{j,k}(x) = \psi(2^j \cdot x - k) \quad \text{(EQ 4.9)}$$

that are generated by the single mother function $\psi(x)$.

If the generating function $\psi(x)$ is assumed to have unit length, meaning that $\|\psi(x)\|_2 = 1$, then the new functions $\psi_{j,k}(x)$ can also be scaled to have unit length by defining:

$$\psi_{j,k}(x) = 2^{j/2} \cdot \psi(2^j \cdot x - k) \qquad \text{(EQ 4.10)}$$

If the set of functions $\{\psi_{j,k}(x)\}$ is an *orthonormal basis* of $L^2(R)$, which means that

$$\langle \psi_{j,k}(x), \psi_{l,m}(x) \rangle = \delta_{j,l} \cdot \delta_{k,m} \qquad \text{(EQ 4.11)}$$

and each function $f(x) \in L^2(R)$ can be written as

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_{j,k} \cdot \psi_{j,k}(x) \qquad \text{(EQ 4.12)}$$

then the generating function $\psi(x)$ is called an *orthonormal wavelet*.

If the set of functions $\{\psi_{j,k}(x) \,|\, j,k \in Z\}$ is a *Riesz basis* of $L^2(R)$ then the generating function $\psi(x) \in L^2(x)$ is called an R*-function*. Furthermore an R-function is also an R-*wavelet* if there exists a function $\tilde{\psi}(x) \in L^2(R)$, a so-called *dual wavelet*, such that:

$$\langle \psi_{j,k}(x), \tilde{\psi}_{l,m}(x) \rangle = \delta_{j,l} \cdot \delta_{k,m} \qquad \text{(EQ 4.13)}$$

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \langle f(x), \psi_{j,k}(x) \rangle \cdot \tilde{\psi}_{j,k}(x) \qquad \text{(EQ 4.14)}$$

Next, we can define the space generated by the closure of the linear span of $\{\psi_{j,k}(x) : k \in Z\}$:

$$W_j = \text{clos}_{L^2(R)} \text{ span } \{\psi_{j,k}(x) : k \in Z\} \qquad \text{(EQ 4.15)}$$

From the definition of a space W_j in Eq. 4.15 and the definition of the function $\psi(x)$ the space $L^2(R)$ can be described as a *direct sum* $\oplus$ of the spaces W_j:

$$L^2(R) = \bigoplus_{j \in Z} W_j \qquad \text{(EQ 4.16)}$$

One last component needed for a multiresolution analysis is the definition of the space V_j:

$$V_j = \ldots \oplus W_{j-2} \oplus W_{j-1} \qquad \text{(EQ 4.17)}$$

Definition 4: *Any R-wavelet (or wavelet) generates a direct sum decomposition of $L^2(R)$ described in Eq. 4.16. Equivalently any wavelet builds a multiresolution approximation of $L^2(R)$ with the following properties:*

1. $\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset \ldots$
2. $\bigcap_{j=-\infty}^{\infty} V_j = \{\varnothing\}$ *and* $\bigcup_{j=-\infty}^{\infty} V_j$ *is dense in* $L^2(R)$
3. $f(x) \in V_j \Leftrightarrow f(2 \cdot x) \in V_{j+1}$, *for all* $f(x) \in L^2(R)$ *and all* $j \in Z$
4. $f(x) \in V_0 \Leftrightarrow f(x-k) \in V_0$ *for all* $f(x) \in L^2(R)$ *and all* $k \in Z$
5. $V_{j+1} = V_j \oplus W_j$, *for all* $j \in Z$
6. *There exists a mother function* $\phi(x) \in L^2(R)$ *that defines a set of scalar basis functions* $\phi_{j,k}(x) = 2^{j/2} \cdot \phi(2^j \cdot x - k)$ *whose closure generates* V_j*:*

$$V_j = \text{clos}_{L^2(R)} \text{ span } \{\phi_{j,k}(x): k \in Z\} \qquad \text{(EQ 4.18)}$$

One final comment: the scalar and wavelet basis functions defined in this section satisfy the *two-scale relation*, which is used to construct the basis functions at level j using a linear combination of the basis functions at level $j+1$:

$$\phi_{j,l}(x) = \sum_{k=-\infty}^{\infty} p_k \cdot \phi_{j+1,k}(x) \qquad \text{(EQ 4.19)}$$

$$\psi_{j,l}(x) = \sum_{k=-\infty}^{\infty} q_k \cdot \phi_{j+1,k}(x) \qquad \text{(EQ 4.20)}$$

4.1.3 Filter-Bank Representation of Signals

As discussed in the previous sections the wavelet theory is used to represent any function $f(x) \in V_i$ in $L^2(R)$ at different levels of resolution. This is accomplished by projecting the function $f(x)$ in the subspace V_{i-1}, and by storing the difference in the orthogonal subspace W_{i-1}. This process can be repeated n times, resulting in a multiresolution representation of the function $f(x)$.

The decomposition and reconstruction operations can be defined with four matrices A, B, P, and Q as follows:

- The two analysis matrices A and B decompose a discrete signal $\boldsymbol{c_i} \in V_i$ into a coarser discrete signal $\boldsymbol{c_{i-1}} \in V_{i-1}$ and a difference signal $\boldsymbol{d_{i-1}} \in V_{i-1}$:

$$\boldsymbol{c_{i-1}} = A \cdot \boldsymbol{c_i} \qquad \text{(EQ 4.21)}$$

$$\boldsymbol{d_{i-1}} = B \cdot \boldsymbol{c_i} \qquad \text{(EQ 4.22)}$$

- Two synthesis matrices P and Q reconstruct the discrete signal $\boldsymbol{c_i} \in V_i$ from the coarse signal $\boldsymbol{c_{i-1}} \in V_{i-1}$ and the difference signal $\boldsymbol{d_{i-1}} \in W_{i-1}$:

$$\boldsymbol{c_i} = P \cdot \boldsymbol{c_{i-1}} + Q \cdot \boldsymbol{d_{i-1}} \qquad \text{(EQ 4.23)}$$

These two steps can be repeated on the input signal, thus generating a multiresolution representation of the signal. The resulting algorithm, called *filter bank algorithm*, is illustrated in Figure 4.1 for a discrete input signal $\boldsymbol{c_i} \in V_i$:

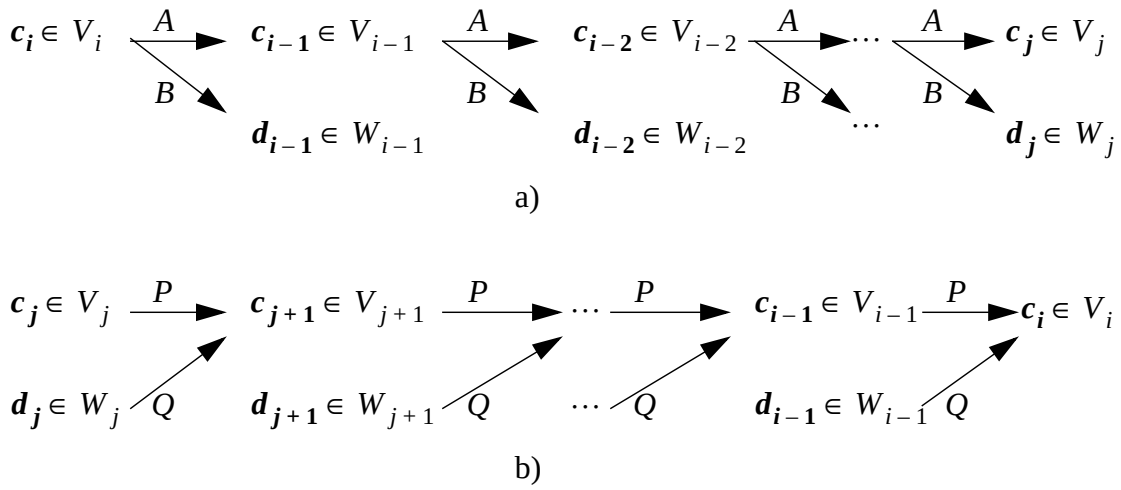


Figure 4.1 Filter bank algorithm
a) Decomposition step
b) Reconstruction step

After the decomposition step the input signal $\boldsymbol{c_i} \in V_i$ is represented by its projection $\boldsymbol{c_j}$ onto the space V_j plus all the discrete details signals $\{\boldsymbol{d_k} \in V_k \,|\, k = j, \ldots, i-1\}$ needed to reconstruct the signal.

The matrices P and Q used in the reconstruction step can be derived from the two-scale relations expressed in Eq. 4.19 and Eq. 4.20, and the projection matrices used in the decomposition step can be constructed using Eq. 4.24:

$$\begin{pmatrix} A \\ B \end{pmatrix} = (P, Q)^{-1} \quad \text{(EQ 4.24)}$$

Let $\dim(\boldsymbol{c_i})$ denote the size of the signal $\boldsymbol{c_i} \in V_i$. The size of the matrix P which upsamples the signal $\boldsymbol{c_i} \in V_i$ to the space V_{i+1} has the size $\dim(\boldsymbol{c_{i+1}}) \times \dim(\boldsymbol{c_i})$, the matrix Q, which adds the details $\boldsymbol{d_i} \in W_i$ back to reconstruct $\boldsymbol{c_{i+1}} \in V_{i+1}$ has the size $\dim(\boldsymbol{c_{i+1}}) \times (\dim(\boldsymbol{c_{i+1}}) - \dim(\boldsymbol{c_i}))$.

The size of the matrix A, which projects the signal $\boldsymbol{c_{i+1}} \in V_{i+1}$ to $\boldsymbol{c_i} \in V_i$ has the size $\dim(\boldsymbol{c_i}) \times \dim(\boldsymbol{c_{i+1}})$, and the matrix B which is used to compute the detail signal $d_i \in V_i$ has the size $(\dim(\boldsymbol{c_{i+1}}) - \dim(\boldsymbol{c_i})) \times \dim(\boldsymbol{c_{i+1}})$.

4.1.4 Orthogonality of Wavelets

In the literature many different scalar and wavelet basis functions have been constructed that satisfy the conditions described in Section 4.1.2; some of the better known basis functions will be described in Section 4.1.5. These bases can be classified according to their degree of orthogonality:

- *Orthogonal wavelets*: full-orthogonality is the tightest constraint for a basis function, which must satisfy

  $$\langle \psi_{j,k}(x), \psi_{l,m}(x) \rangle = \delta_{j,l} \cdot \delta_{k,m} \quad \text{(EQ 4.25)}$$

  Eq. 4.25 implies that the inner product between a wavelet basis and a scaled and/or translated wavelet basis must be equal to zero.

- *Semi-orthogonal wavelets*: semi-orthogonality is a more relaxed constraint, which requires a basis function to satisfy

  $$\langle \psi_{j,k}(x), \psi_{l,m}(x) \rangle = \delta_{j,l} \quad \text{(EQ 4.26)}$$

  Eq. 4.26 implies that the inner product of two wavelet basis at different scales must be equal to zero.

- *Bi-orthogonal wavelets*: if a wavelet basis does not satisfy the semi-orthogonality condition, then it must satisfy the bi-orthogonality condition

  $$\langle \psi_{j,k}(x), \tilde{\psi}_{l,m}(x) \rangle = \delta_{j,l} \cdot \delta_{k,m} \quad \text{(EQ 4.27)}$$

  Eq. 4.27 implies that the inner product between the primal wavelet basis $\psi_{j,k}(x)$ and the dual wavelet basis $\tilde{\psi}_{l,m}(x)$ at different scales and/or for different translations must be equal to zero.

The orthogonality conditions for the scalar basis functions follow directly from Eq. 4.25 through Eq. 4.27.

4.1.5 Bases for Tensor Product Wavelets

The multiresolution analysis described in Section 4.1.2 did not specify which functions $\phi(x)$, the so called *scalar functions*, and $\psi(x)$, the so called *wavelet functions*, are valid generating functions. In this section we briefly examine some well known basis functions:

- *B-Spline wavelet*s: are constructed from the B-Spline functions, and they represent one of the better known family of wavelets. The scalar and wavelet functions are defined as:

$$\phi_{j,k}(x) = N^m(2^j \cdot x - k) \quad \text{(EQ 4.28)}$$

$$\psi_{j,k}(x) = \sum_{l=0}^{3 \cdot m - 2} q_l \cdot \phi_{j+1, k+l}(x) \quad \text{(EQ 4.29)}$$

$$q_l = (-1)^l \cdot 2^{1-m} \cdot \sum_{n=0}^{m} \binom{m}{n} \cdot N^m(l + 1 - n) \quad \text{(EQ 4.30)}$$

where the $N^m(x)$ is the B-Spline function defined by:

$$N_i^1(u) = \begin{cases} 1 \text{ if } u_i \le u < u_{i+1} \\ 0 \text{ otherwise} \end{cases} \quad \text{(EQ 4.31)}$$

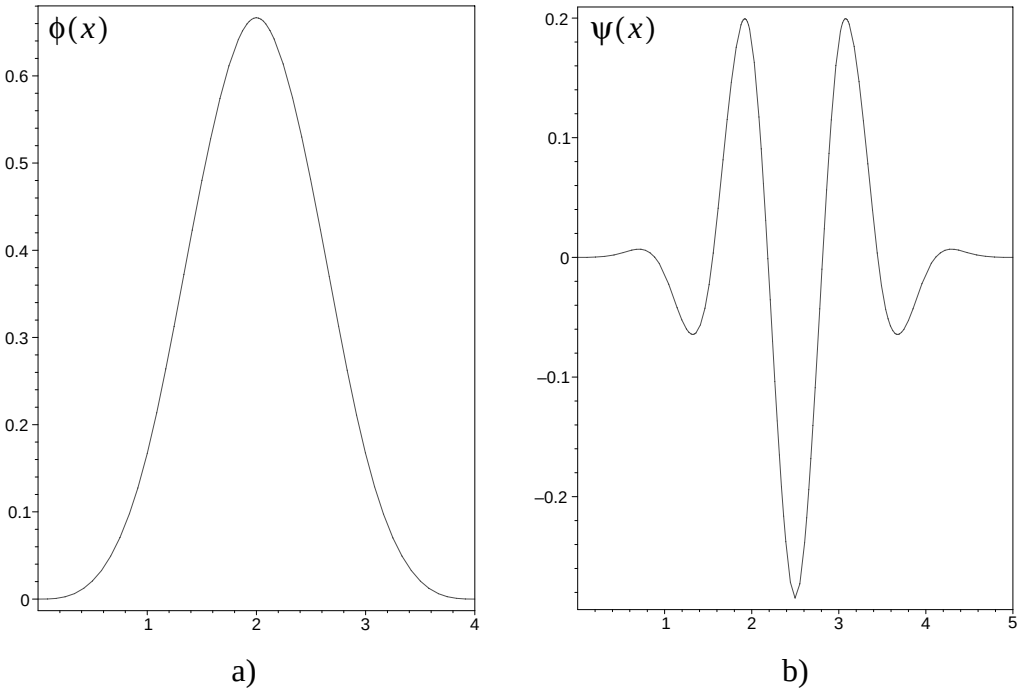
$$N_i^m(u) = \frac{u - u_i}{u_{i+m} - u_i} N_i^{m-1}(u) + \frac{u_{i+m+1} - u}{u_{i+m+1} - u_{i+1}} N_{i+1}^{m-1}(u) \quad \text{(EQ 4.32)}$$

and m specifies the order of the B-Spline basis function. The set of parameters $\{u_i | i \in Z\}$ define a partition of the unit interval where the B-Spline basis functions are defined.

The most important properties of B-Spline wavelets are listed below:

1. They are symmetric for even m and antisymmetric for odd m
2. They build a semi-orthogonal basis
3. They have $m - 1$ vanishing moments

The scalar and wavelet basis functions generated by a cubic B-Spline are drawn in Figure 4.2.



a) b)

Figure 4.2 Cubic B-Splines wavelets
a) Scalar basis function
b) Wavelet basis function

The wavelet transform for discrete surfaces computed with the B-Spline basis functions is implemented with matrix operations. The matrix-matrix and matrix-vector multiplication operations needed to compute the wavelet transform are very expensive: they have a complexity of $O(n^2)$, where n is the number of points on the surface. In order to compute the wavelet transform faster linear algebra approaches are necessary to reduce the total complexity of these operations. Using these approaches the complexity can be reduced to $O(n)$.

More information on the endpoint interpolating B-Spline basis functions can be found in [68], [11], and [69].

- *Battle-Lemarie wavelets*: these basis functions were constructed independently by Battle in [2] and Lemarie in [52]. The Battle-Lemarie scalar and basis functions orthogonalize the semi-orthogonal scalar and wavelet B-Spline basis functions.
  This goal is achieved by use of a theorem that states that the family $\{\phi(x-k): k \in Z\}$ is an orthonormal basis if and only if the following equation holds:

$$\sum_{k=-\infty}^{\infty} |\Phi(\omega + 2\pi k)|^2 = 1 \qquad \text{(EQ 4.33)}$$

where $\Phi(\omega)$ is the fourier transformed of the basis function $\phi(x)$.

This theorem can be applied directly to define the Fourier transform $\Phi(\omega)$ of the new scaling function as

$$\Phi(\omega) = \frac{\hat{N}^m(\omega)}{\left(\sum_{k=-\infty}^{\infty} \left| \hat{N}^m(\omega + 2\pi k) \right|^2 \right)} \quad \text{(EQ 4.34)}$$

where $\hat{N}^m(\omega)$ is the Fourier transform of the B-Spline basis defined in Eq. 4.31 and Eq. 4.32.
The Fourier transform of the new wavelet function $\Psi(\omega)$ is defined as

$$\Psi(\omega) = -e^{-i \cdot \frac{\omega}{2}} \cdot \frac{\overline{\Phi(\omega + 2\pi)}}{\overline{\Phi\left(\frac{\omega}{2} + \pi\right)}} \Phi\left(\frac{\omega}{2}\right) \quad \text{(EQ 4.35)}$$

The formula for the scalar and wavelet basis functions can be defined as:

$$\phi(x) = \sum_{n=-\infty}^{\infty} c_n \cdot N^m(x - n) \quad \text{(EQ 4.36)}$$

$$\psi(x) = \sum_{n=-\infty}^{\infty} (d_{n+1} - 2d_n + d_{n-1}) \cdot \phi(2x - n) \quad \text{(EQ 4.37)}$$

The coefficients c_n for the scalar basis function Eq. 4.36 correspond to the Fourier coefficients of Eq. 4.34, and the coefficients d_n for the wavelet basis function in Eq. 4.37 correspond to the Fourier coefficients of Eq. 4.35.

Figure 4.3 illustrates the Battle-Lemarie scalar and wavelet basis function obtained from the cubic B-Spline basis functions via the orthogonalization process:

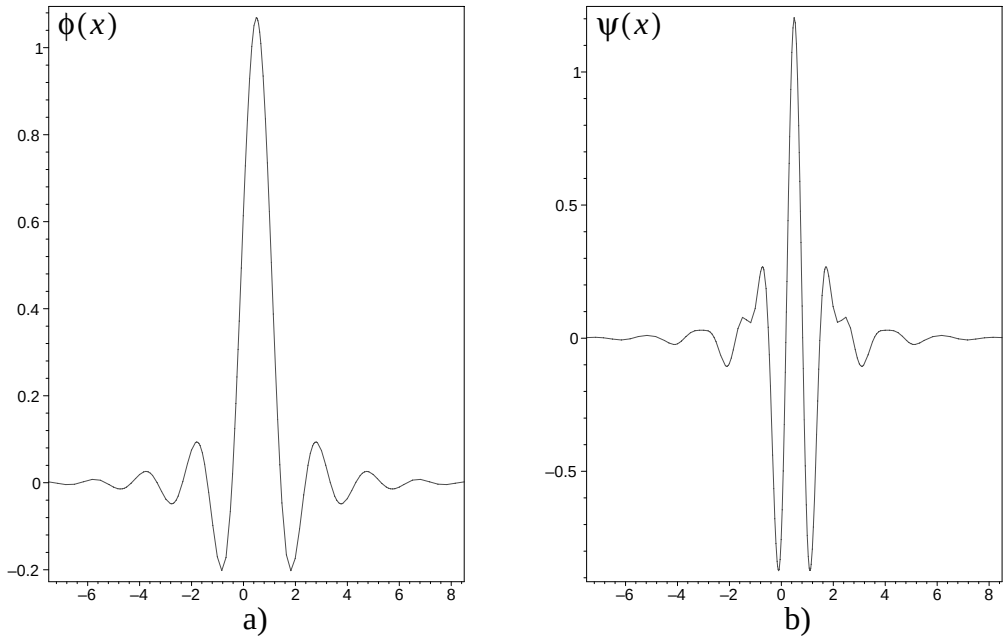


Figure 4.3 Battle Lemarie wavelets generated from the cubic B-Spline function
a) Scalar basis function
b) Wavelet basis function

The orthogonalization process described in this section that allows to build the Battle-Lemarie wavelets has the drawback of generating wavelets without a local support: the scalar and wavelet basis functions shown in Figure 4.3 decay exponentially toward zero, and their support is the whole real line. As a result the matrices P, Q, A, and B are dense, and the complexity of the decomposition and reconstruction steps of a filter bank algorithm is therefore quadratic in the number of points in the surface.

A detailed description of the Battle-Lemarie wavelets can be found in the original papers of Battle [2] and Lemarie [52], as well as in [10] and [16].

- *Gabor wavelets*: the Gabor wavelets have been primarily adopted in image processing applications as an analysis tool. One important feature of Gabor wavelets is they mimic the behavior of pairs of cells in the primary visual cortex well. The wavelet transform is not computed with a tensor product ansatz using one-dimensional basis functions as in the case of the other wavelet bases described in this section, but by convolution:

$$r(x) = \int_{-\infty}^{\infty} f(x)\psi(x' - x)dx' \qquad \text{(EQ 4.38)}$$

The scalar function is defined as a Gaussian curve

$$\phi(x) = \frac{1}{\sqrt{\pi}\sigma} e^{-\frac{x^T x}{2\sigma^2}} \qquad \text{(EQ 4.39)}$$

whereas the wavelet function is modeled as a plane wave weighted by a Gaussian function:

$$\psi(x) = \frac{1}{\sqrt{\pi}\sigma} e^{irx} e^{-\frac{x^T x}{2\sigma^2}} \qquad \text{(EQ 4.40)}$$

Furthermore, the plane wave can be oriented, so that many different wavelet functions are generated.

Note that the Gabor wavelets do not have vanishing moments.

Figure 4.4 illustrates an example of Gabor scalar and wavelet functions:

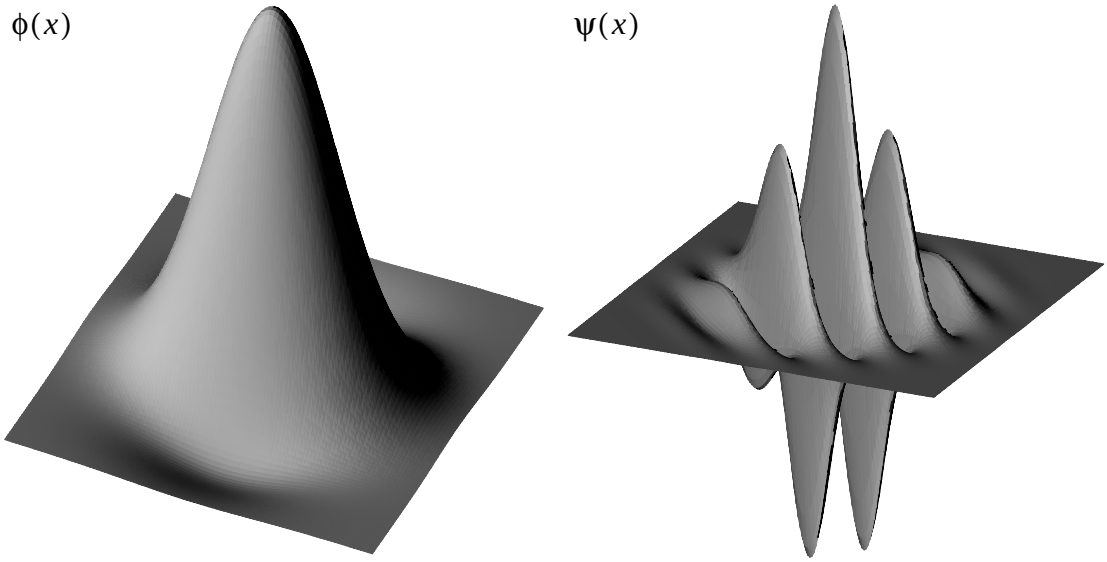


Figure 4.4 Gabor wavelets
a) Scalar basis function
b) Wavelet basis function

More information on Gabor wavelets and filters can be found in [38], [3] and [20].

- *Daubechies wavelets*: the Daubechies scalar and wavelet functions do not have a closed form; the bases are implicitly computed, for example, by spectral factorization of a trigonometric polynomial. As an example, we show how to construct an orthonormal basis with a compact support and maximum number of vanishing moments.

  The first step consists in finding the non-vanishing coefficients h_n of

$$m_0(\omega) = \frac{1}{\sqrt{2}} \cdot \sum_{n=0}^{2N-1} h_n \cdot e^{-in\omega} \qquad \text{(EQ 4.41)}$$

  The function $m_0(\omega)$ is used to define a two-scale relation-like formula for the scalar basis function in Fourier space:

$$\Phi(\omega) = m_0\left(\frac{\omega}{2}\right) \cdot \Phi\left(\frac{\omega}{2}\right) \tag{EQ 4.42}$$

The coefficients can be computed using the assumption that

$$m_0(\omega) = \left(\frac{1 + e^{-i\omega}}{2}\right)^N \cdot S(\omega) \tag{EQ 4.43}$$

where

$$\begin{aligned} \mathrm{L}(\omega) = |S(\omega)|^2 = P\left(\sin\left(\frac{\omega}{2}\right)^2\right) = \\ = \sum_{k=0}^{N-1} \binom{N-1+k}{k} \sin\left(\frac{\omega}{2}\right)^{2k} + \sin^N\left(\frac{\omega}{2}\right)^{2N} R\left(\frac{1}{2} - \sin\left(\frac{\omega}{2}\right)^2\right) \end{aligned} \tag{EQ 4.44}$$

In order to generate a set of orthonormal basis functions with minimal support and maximal number of vanishing moments Eq. 4.45 must hold:

$$R\left(\frac{1}{2} - \sin\left(\frac{\omega}{2}\right)^2\right) \equiv 0 \tag{EQ 4.45}$$

The coefficient h_n can then be computed by evaluating $\mathrm{L}(\omega)$ and representing it as

$$\mathrm{L}(\omega) = \sum_{m=0}^{M} l_m \cdot \cos(m\omega) \tag{EQ 4.46}$$

and then using a *Lemma of Riesz*, which states that for each trigonometric polynomial of the form Eq. 4.46 there exists a trigonometric polynomial of the form

$$B(\omega) = \sum_{m=0}^{M} b_m \cdot e^{-im\omega} \tag{EQ 4.47}$$

such that $|B(\omega)|^2 = \mathrm{L}(\omega)$.

Once the coefficients h_n have been computed it is possible to compute the scalar and wavelet bases using the two scale relation:

$$\begin{aligned} \phi(x) &= \sqrt{2}\sum_n h_n \phi(2x - n) \\ \psi(x) &= \sqrt{2}\sum_n (-1)^n h_{-n+1} \phi(2x - n) \end{aligned} \tag{EQ 4.48}$$

Finally, it is necessary to compute the value of $\phi(x)$ at a finite number of positions, for example at all integer positions, and then use the two scale relation formula to compute the value of the function at any other position.

Figure 4.5 shows some basis functions generated using this scheme:

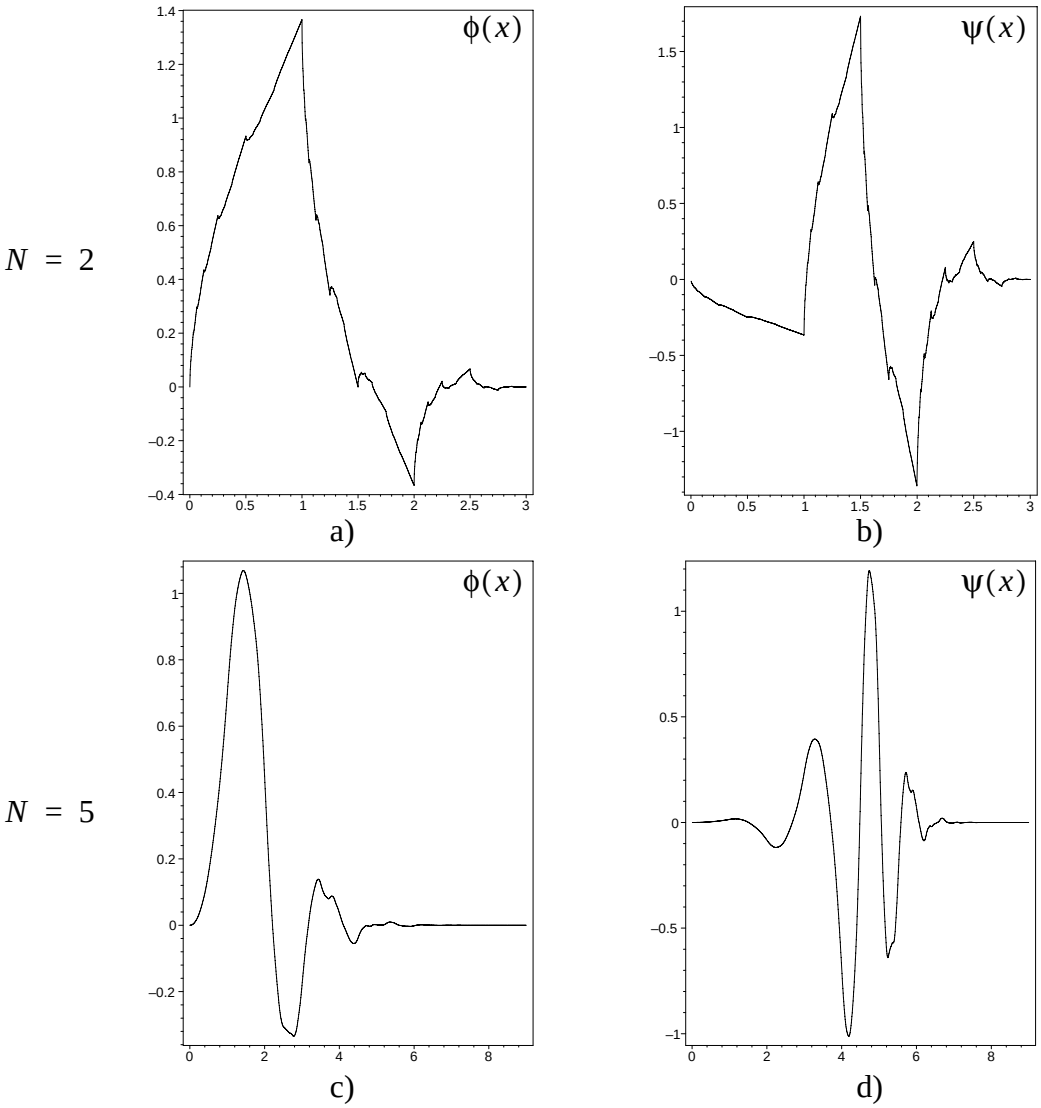


Figure 4.5 Daubechies wavelets
a) Scalar basis function for *N=2*
b) Wavelet basis function for *N=2*
c) Scalar basis function for *N=5*
d) Wavelet basis function for *N=5*

The scheme we presented in this section constructs wavelet bases with a maximum number of vanishing moments. The scheme generates a subset of the family of the Daubechies wavelet functions. More information on this topic can be found in [16], [15], [17], and [13].

4.1.6 How to Represent a Grid with First Generation Wavelets

In this section we show how to construct a multiresolution representation of a two dimensional regular grid using the simple Haar basis function defined as:

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{(EQ 4.49)}$$

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \le x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le x < 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{(EQ 4.50)}$$

A wavelet representation for a multidimensional grid can be computed easily using the tensor product approach: a one dimensional wavelet transform is applied at each dimension, one at a time.

Figure 4.6 shows the decomposition step of a two-dimensional image:

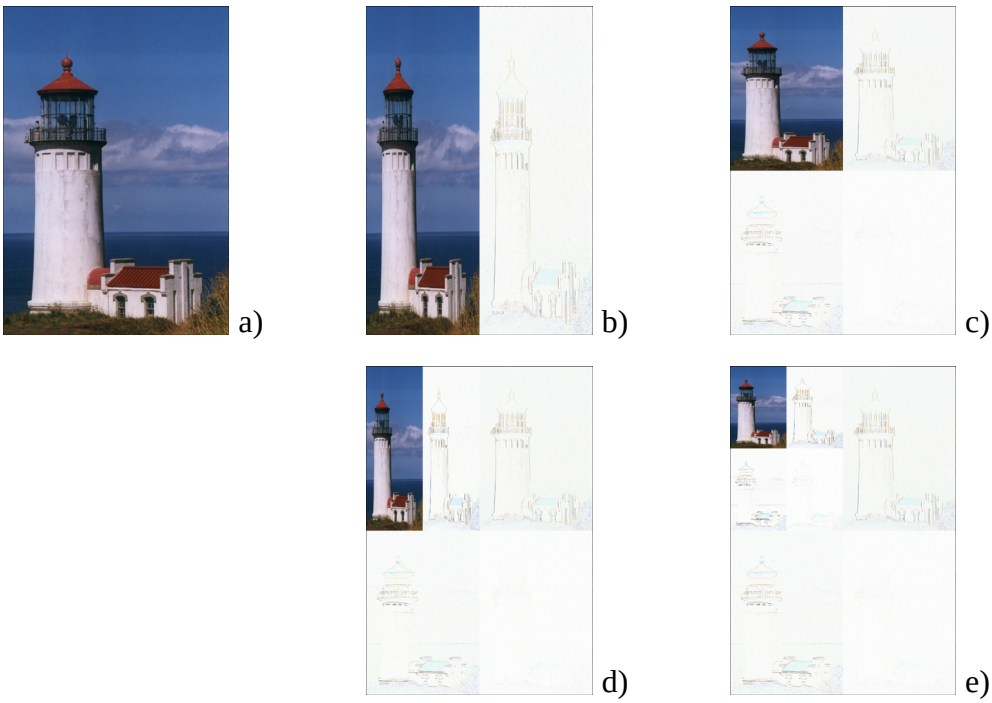


Figure 4.6 Wavelet decomposition of an image (magnitude of the wavelet details increased by 4)
a) Original input data
b) One dimensional wavelet transform in *x*
c) One dimensional wavelet transform in *y*; first level of decomposition
d) One dimensional wavelet transform in *x*
e) One dimensional wavelet transform in *y*; second level of decomposition

One level of decomposition of an input two dimensional grid (Figure 4.6-a) is achieved by first computing a wavelet transform along one dimension (Figure 4.6-b), and then along the other dimension (Figure 4.6-c).
Using the terminology developed in Section 4.1.3 the input grid can be thought as a dis-

crete signal $\boldsymbol{c_{i,i}} \in V_i \times V_i$. The first wavelet transform decomposes one of the dimensions, which results in two new signals:

$$\boldsymbol{c_{i-1,i}} = A \cdot \boldsymbol{c_{i,i}} \in V_{i-1} \times V_i \quad \text{(EQ 4.51)}$$

$$\boldsymbol{d_{i-1,i}} = B \cdot \boldsymbol{c_{i,i}} \in W_{i-1} \times V_i \quad \text{(EQ 4.52)}$$

In Figure 4.6-b $\boldsymbol{c_{i-1,i}}$ is represented by the upper-left half of the surface, which retains many of the features of the original surface, and $\boldsymbol{d_{i-1,i}}$ is represented by the other half, which is almost flat.
After applying the wavelet in the second dimension, the surface is divided into four subpatches: $\boldsymbol{c_{i-1,i}}$ is decomposed in

$$\boldsymbol{c_{i-1,i-1}} = A \cdot \boldsymbol{c_{i-1,i}} \in V_{i-1} \times V_{i-1} \quad \text{(EQ 4.53)}$$

$$\boldsymbol{d^1_{i-1,i-1}} = B \cdot \boldsymbol{c_{i-1,i}} \in V_{i-1} \times W_{i-1} \quad \text{(EQ 4.54)}$$

and $\boldsymbol{d_{i-1,i}}$ is decomposed in

$$\boldsymbol{d^2_{i-1,i-1}} = A \cdot \boldsymbol{d_{i-1,i}} \in W_{i-1} \times V_{i-1} \quad \text{(EQ 4.55)}$$

$$\boldsymbol{d^3_{i-1,i-1}} = B \cdot \boldsymbol{d_{i-1,i}} \in W_{i-1} \times W_{i-1} \quad \text{(EQ 4.56)}$$

The subpatch $\boldsymbol{c_{i-1,i-1}}$, which contains the coarser representation of the surface, is then further decomposed in Figure 4.6 d-e.

4.2 Second Generation Wavelets

As we have seen in the previous section, classic tensor-product wavelets that span $L^2(R)$ are generated through binary dilation and dyadic translation. This type of basis functions is more than adequate to represent functions $f(x)$ defined over $L^2(R)$, but it is not general and powerful enough to handle irregular settings, such as a function defined on a sphere.

In second generation wavelets, basis functions are generated using the lifting scheme, which allows to construct a broader range of bases than the dilation and translation operators in the first generation wavelets: the construction starts with a choice of a simple wavelet, which is lifted to a new and more complex wavelet that possesses a set of properties useful to model specific types of signals. The lifting scheme allows to generate first generation wavelets, as well as new types of wavelets defined over more complicated domains, such as spherical domains or quaternary subdivision meshes.

In the first subsection we will introduce the basics of the new multiresolution analysis for the second generation wavelet; next we will present the lifting theorem, and finally we will give a simple examples of how the lifting process works.

More detail on second generation wavelets and their related theory can be found in [81], [82], [83], [84], [33], [19], [73], and [85].

4.2.1 Multiresolution Analysis of Second Generation Wavelets

Second generation wavelets generalize the wavelet theory to more general settings than $L^2(R)$. Ideally this new generation of wavelets should preserve some of the properties of the first generation wavelets, such as:

1. The wavelets must form a Riesz basis of $L^2(R)$ (see Section 4.1) as well as an unconditional basis for a space F, so that a function $f(x) \in F$ can be represented as

$$f(x) = \sum_{j=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \gamma_{j,m} \cdot \psi_{j,m}(x) \qquad \text{(EQ 4.57)}$$

   where $\{\psi_{j,m}(x) | j, m \in Z\}$ is the set of wavelet basis functions and $\{\gamma_{j,m}(x) | j, m \in Z\}$ are the coefficients of the linear combination.

2. The wavelets are either orthogonal or the dual wavelets are known.
3. The wavelets and the dual wavelets are local both in space and frequency.
4. The wavelets fit into a multiresolution analysis framework.

In addition to this basic set of properties the second generation wavelets are expected to fulfill the following properties as well:

1. It should be possible to define wavelets on more general settings than R^n, like wavelets on curves, surfaces and more generally on manifolds.
2. It would be desirable to define wavelets over irregularly sampled data, since the data acquired in real life is often irregularly sampled.

Now that the goals of this new representation have been set, we will begin the investigation of the second generation wavelets by constructing a multiresolution analysis. The multiresolution analysis is built on a space $L^2 = L^2(X, \Sigma, \mu)$, where $X \subset R^n$ is the spatial domain, Σ is a σ-algebra and μ is a non-atomic measure on Σ. Then:

Definition 5: *A multiresolution analysis M of L^2 is a sequence of subspaces $M = \{V_j \subset L^2 | j \in J \subset Z\}$ such that*

1. $\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset \ldots$

2. $\bigcap_{j \in J} V_j = \{\varnothing\}$ *and* $\bigcup_{j \in J} V_j$ *is dense in* L^2
3. V_j *has a Riesz basis defined by* $\{\phi_{j,k}(x) \mid k \in K(j)\}$, *where* $K(j)$ *is an index set*

The lifting scheme does not lift orthogonal or semi-orthogonal bases (see Section 4.1.4) in general, but only bi-orthogonal bases. Therefore it is necessary to define a dual *multiresolution analysis* $\tilde{M} = \{\tilde{V}_j \subset L^2 \mid j \in J\}$. The spaces $\tilde{V}_j$ have a Riesz basis defined by $\{\tilde{\phi}_{j,k} \mid k \in \tilde{K}(j)\}$.

The construction of the scaling functions $\phi_{j,k}(x)$ and their duals $\tilde{\phi}_{j,k}(x)$ are computed using the two-scale relation formula:

$$\phi_{j,k}(x) = \sum_{l \in K(j+1)} h_{j,k,l} \cdot \phi_{j+1,l}(x) \qquad \text{(EQ 4.58)}$$

$$\tilde{\phi}_{j,k}(x) = \sum_{l \in \tilde{K}(j+1)} \tilde{h}_{j,k,l} \cdot \tilde{\phi}_{j+1,l}(x) \qquad \text{(EQ 4.59)}$$

where $\{h_{j,k,l} \mid l \in K(j+1)\}$ and $\{\tilde{h}_{j,k,l} \mid l \in \tilde{K}(j+1)\}$ are the sets of coefficients of the two linear combinations. A more detailed description of this construction can be found in [81].

Definition 5 specified a multiresolution analysis for this new setting and defined the scalar basis functions. Definition 6 describes the wavelet basis functions:

Definition 6: *A set of functions* $\{\psi_{j,m}(x) \mid j \in J, m \in M(j)\}$, *where* $M(j)$ *is defined as* $M(j) = K(j+1) \setminus K(j)$, *is a set of wavelet functions if:*

1. *The space* $W_j = \mathrm{clos}_{L^2}\,\mathrm{span}\{\psi_{j,m}(x) \mid m \in M(j)\}$ *is a complement of* V_j *in* V_{j+1} *and* $W_j \perp \tilde{V}_j$.
2. $\left\{ \frac{\psi_{j,m}(x)}{\|\psi_{j,m}(x)\|} \,\middle|\, j \in J = N, m \in M(j) \right\} \cup \left\{ \frac{\phi_{0,k}(x)}{\|\phi_{0,k}(x)\|} \,\middle|\, k \in K(0) \right\}$ *or* $\left\{ \frac{\psi_{j,m}(x)}{\|\psi_{j,m}(x)\|} \,\middle|\, j \in J = Z, m \in M(j) \right\}$ *form a Riesz basis for* L^2.

As already mentioned the lifting scheme will not lift orthogonal or semi-orthogonal bases in general, but only bi-orthogonal bases. It is therefore necessary to define a complement

space $\tilde{W}_j$ and a complement set of wavelet basis functions $\{\tilde{\psi}_{j,m}(x) \mid j \in J, m \in \tilde{M}(j)\}$. The new space $\tilde{W}_j$ is the complement of $\tilde{V}_j$ in $\tilde{V}_{j+1}$, and it is orthogonal to V_j.

The primal and dual wavelet basis functions must also satisfy the two-scale relation defined in Eq. 4.60

$$\psi_{j,m}(x) = \sum_{l \in K(j+1)} g_{j,m,l} \cdot \phi_{j+1,l}(x) \quad \text{(EQ 4.60)}$$

$$\tilde{\psi}_{j,m}(x) = \sum_{l \in \tilde{K}(j+1)} \tilde{g}_{j,m,l} \cdot \tilde{\phi}_{j+1,l}(x) \quad \text{(EQ 4.61)}$$

where $\{g_{j,m,l} \mid l \in K(j+1)\}$ and $\{\tilde{g}_{j,m,l} \mid l \in \tilde{K}(j+1)\}$ are the sets of coefficients of the two linear combinations.

The orthogonality conditions that a wavelet basis function must meet are described in Section 4.1.4.

4.2.2 The Lifting Scheme

As we mentioned in the introduction of this section, the core component of second generation wavelets is the lifting step. A lifting step can either lift the dual scalar function and the primal wavelet function or the primal scalar function and the dual wavelet function. These two lifts will be introduced in Theorem 1 and in Theorem 2. Figure 4.7 and Figure 4.8 show how the lifting steps can be implemented into a fast wavelet transform.

Theorem 1 describes how to lift the dual scalar basis function and the primal wavelet basis function:

Theorem 1: *From an initial set of biorthogonal basis functions* $\{\phi^{\text{old}}(x), \tilde{\phi}^{\text{old}}(x), \psi^{\text{old}}(x), \tilde{\psi}^{\text{old}}(x)\}$ *a new set of biorthogonal basis functions* $\{\phi(x), \tilde{\phi}(x), \psi(x), \tilde{\psi}(x)\}$ *can be constructed as:*

$$\phi_{j,k}(x) = \phi^{\text{old}}_{j,k}(x) \quad \text{(EQ 4.62)}$$

$$\tilde{\phi}_{j,k}(x) = \sum_{l \in \tilde{K}(j+1)} \tilde{h}^{\text{old}}_{j,k,l} \cdot \tilde{\phi}_{j+1,l}(x) + \sum_{m \in \tilde{M}(j)} s_{j,k,m} \cdot \tilde{\psi}_{j,m}(x) \quad \text{(EQ 4.63)}$$

$$\psi_{j,m}(x) = \psi^{\text{old}}_{j,m}(x) - \sum_{k \in K(j)} s_{j,k,m} \cdot \phi_{j,m}(x) \quad \text{(EQ 4.64)}$$

$$\tilde{\psi}_{j,m}(x) = \sum_{l \in \tilde{K}(j+1)} \tilde{g}^{\text{old}}_{j,m,l} \cdot \tilde{\phi}_{j+1,l}(x) \quad \text{(EQ 4.65)}$$

Theorem 1 can be easily implemented using a fast wavelet transform. Figure 4.7 shows how to construct the decomposition and reconstruction steps of the filter bank algorithm presented in Section 4.1.3 for an input signal $\{c_{j+1,k} | k \in K(j)\}$.

Forward transform from level $j+1$ to level j:

$$\forall k \in K(j): \; c_{j,k} = \sum_{l \in K(j+1)} \tilde{h}^{\text{old}}_{j,k,l} \cdot c_{j+1,l}$$

$$\forall m \in M(j): \; d_{j,k} = \sum_{l \in K(j+1)} \tilde{g}^{\text{old}}_{j,m,l} \cdot c_{j+1,l}$$

$$\forall k \in K(j): \; c_{j,k} = c_{j,k} + \sum_{m \in M(j)} s_{j,k,m} \cdot d_{j,m}$$

Inverse transform from level $j+1$ to level j:

$$\forall k \in K(j): \; c_{j,k} = c_{j,k} - \sum_{m \in M(j)} s_{j,k,m} \cdot d_{j,m}$$

$$\forall l \in K(j+1): \; c_{j+1,l} = \sum_{k \in K(j)} h^{\text{old}}_{j,k,l} \cdot c_{j,k} + \sum_{m \in M(j)} g^{\text{old}}_{j,m,l} \cdot d_{j,m}$$

Figure 4.7 Lifting of the primal wavelet basis functions

Theorem 2 describes how to lift the primal scalar basis function and the dual wavelet basis function:

Theorem 2: *From an initial set of biorthogonal basis functions* $\{\phi^{\text{old}}(x), \tilde{\phi}^{\text{old}}(x), \psi^{\text{old}}(x), \tilde{\psi}^{\text{old}}(x)\}$ *a new set of biorthogonal basis functions* $\{\phi(x), \tilde{\phi}(x), \psi(x), \tilde{\psi}(x)\}$ *can be constructed as:*

$$\phi_{j,k}(x) = \sum_{l \in K(j+1)} h^{\text{old}}_{j,k,l} \cdot \phi_{j+1,l}(x) + \sum_{m \in M(j)} s_{j,k,m} \cdot \psi_{j,m}(x) \quad \text{(EQ 4.66)}$$

$$\tilde{\phi}_{j,k}(x) = \tilde{\phi}^{\text{old}}_{j,k}(x) \quad \text{(EQ 4.67)}$$

$$\psi_{j,m}(x) = \sum_{l \in K(j+1)} g^{\text{old}}_{j,m,l} \cdot \phi_{j+1,l}(x) \quad \text{(EQ 4.68)}$$

$$\tilde{\psi}_{j,m}(x) = \tilde{\psi}^{\text{old}}_{j,m}(x) - \sum_{k \in \tilde{K}(j)} s_{j,k,m} \cdot \tilde{\phi}_{j,m}(x) \quad \text{(EQ 4.69)}$$

Theorem 2 can also be implemented using the fast wavelet transform. The reconstruction step of the filter banks algorithm are shown in Figure 4.8:

Forward transform from level $j+1$ to level j:

$$\forall k \in K(j):\ c_{j,k} = \sum_{l \in K(j+1)} \tilde{h}^{\text{old}}_{j,k,l} \cdot c_{j+1,l}$$

$$\forall m \in M(j):\ d_{j,k} = \sum_{l \in K(j+1)} \tilde{g}^{\text{old}}_{j,m,l} \cdot c_{j+1,l} - \sum_{k \in K(j)} s_{j,k,m} \cdot c_{j,k}$$

Inverse transform from level $j+1$ to level j:

$$\forall k \in K(j):\ d_{j,k} = d_{j,k} + \sum_{k \in K(j)} s_{j,k,m} \cdot c_{j,m}$$

$$\forall l \in K(j+1):\ c_{j+1,l} = \sum_{k \in K(j)} h^{\text{old}}_{j,k,l} \cdot c_{j,k} + \sum_{m \in M(j)} g^{\text{old}}_{j,m,l} \cdot d_{j,m}$$

Figure 4.8 Lifting of the dual wavelet basis functions

From Figure 4.7 and Figure 4.8 we see immediately how simple the implementation of the lifting scheme is: in the forward transform the detail values $\boldsymbol{d_j}$ are modified with a linear combination of the scalar values $\boldsymbol{c_j}$, and the scalar values $\boldsymbol{c_j}$ are then modified with a linear combination of the detail values $\boldsymbol{d_j}$. These two operations can be inverted, and the reconstruction process can be performed efficiently.

The operator $\{s_{j,k,m} | k \in K(j), m \in M(j)\}$ allows to lift a set of basis functions to a new set of functions that possess some new properties. For example, it is possible to choose $\{s_{j,k,m} | k \in K(j), m \in M(j)\}$ so that the p-th vanishing moment becomes zero.

4.2.3 How to Lift the Lazy Wavelet

In this section we show how to lift the lazy wavelet to a biorthogonal wavelet of Cohen-Daubechies-Feauveau with $N = 2$ and $\tilde{N} = 2$. More information on this set of basis functions can be found in [14].

The lifting process is described by the following steps:

1. The lifting process is started by defining the *lazy wavelet*, the simple input wavelet function:

$$c_{-1,k} = c_{0,2k} \qquad \text{(EQ 4.70)}$$

$$d_{-1,k} = c_{0,2k+1} \qquad \text{(EQ 4.71)}$$

An input one-dimensional signal can now be decomposed using the lazy wavelet into a set of scalar values and a set of detail values. Figure 4.9 shows a simple example:

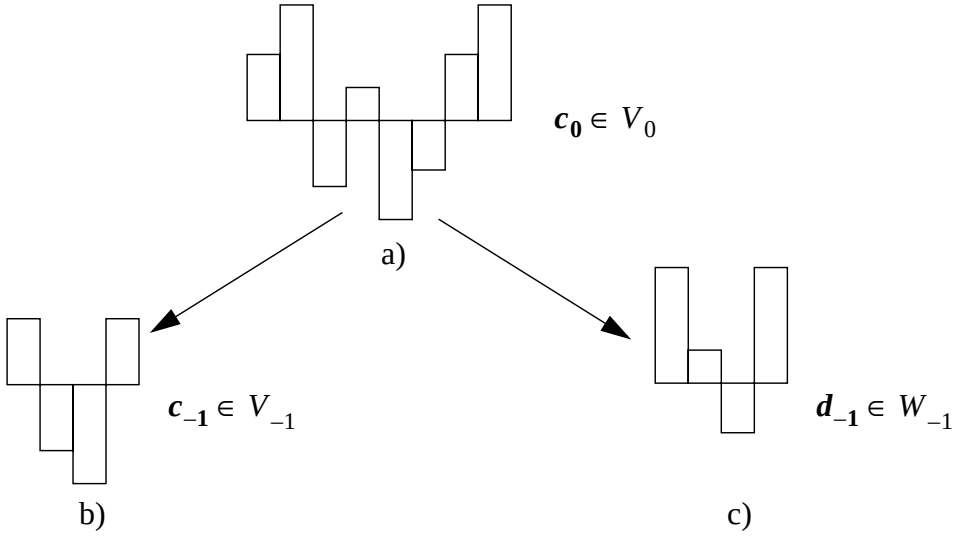


Figure 4.9 One-dimensional input signal decomposed with the lazy wavelet
a) Original input signal
b) Scalar values
d) Detail values

2. The second step consists in finding a good prediction for the detail values $\boldsymbol{d_{-1}}$ using a linear combination of the scalar values $\boldsymbol{c_{-1}}$. As a simple prediction strategy the average of the two scalar values $c_{-1,k}$ and $c_{-1,(k+1) \bmod n}$ can be used:

$$d^{\text{new}}_{-1,k} = d_{-1,k} - \frac{1}{2}(c_{-1,k} + c_{-1,(k+1) \bmod n}) \qquad \text{(EQ 4.72)}$$

The changes in the detail values $\boldsymbol{d_{-1}}$ are illustrated in Figure 4.10:

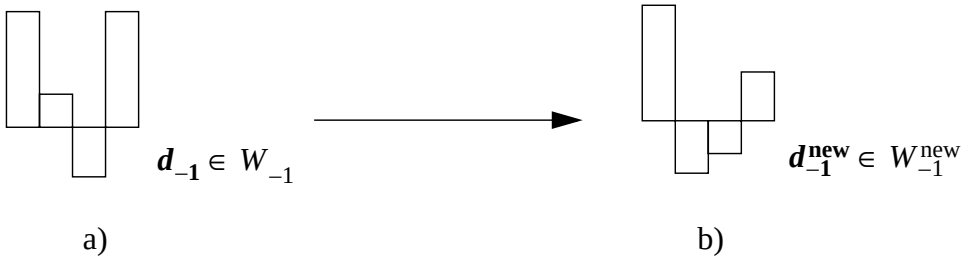


Figure 4.10 Lifting applied to the dual wavelet basis function
a) Detail values before the lift
b) Detail values after the lift

This step corresponds to the lift of the dual wavelet basis defined in Theorem 2.

3. In the last step, the scalar values $\boldsymbol{c_{-1}}$ are scaled to satisfy Eq. 4.73:

$$\sum_k c_{-1,k} = \frac{1}{2}\sum_k c_{0,k} \qquad \text{(EQ 4.73)}$$

This is accomplished by setting

$$c^{\text{new}}_{-1,k} = c_{-1,k} + \frac{1}{4}(d_{-1,(k-1) \bmod n} + d_{-1,k}) \qquad \text{(EQ 4.74)}$$

The changes in the scalar values $\boldsymbol{c_{-1}}$ are illustrated in Figure 4.11:

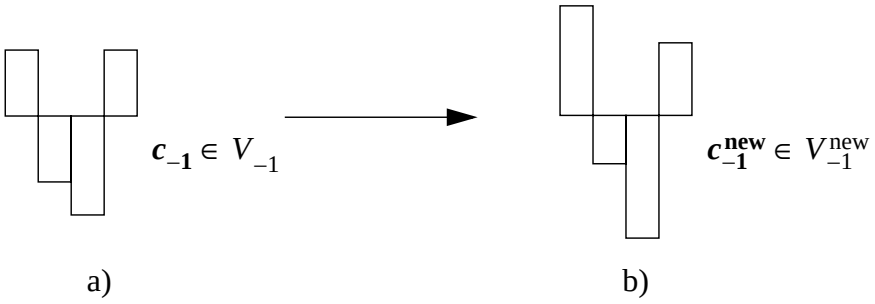


Figure 4.11 Lifting applied to the primal wavelet basis function
a) Scalar values before the lift
b) Scalar values after the lift

This step corresponds to the lift of the primal wavelet basis defined in Theorem 1.

4.3 Multiresolution Analysis with Non-Nested Spaces

George-Pierre Bonneau introduced in [7] and [5] the concept of multiresolution analysis over non-nested spaces, which are generated by the so-called BLaC-wavelets, a combination of the Haar function with the linear B-Spline function.
This concept was then used in [8] and [6] to construct a multiresolution analysis over meshes with arbitrary connectivity.

4.3.1 BLaC Wavelets

The BLaC scalar function is defined as a blending of an Haar basis function and a linear B-Spline basis function, and it is defined as a function which depends on one scalar parameter Δ:

$$\phi^{\Delta}(x) = \begin{cases} \dfrac{x}{\Delta} & \text{if } 0 \le x < \Delta \\ 1 & \text{if } \Delta \le x < 1 \\ 1 - \dfrac{x-1}{\Delta} & \text{if } 1 \le x < 1 + \Delta \\ 0 & \text{otherwise} \end{cases} \qquad \text{(EQ 4.75)}$$

Figure 4.12 shows the BLaC scalar function for different values of Δ:

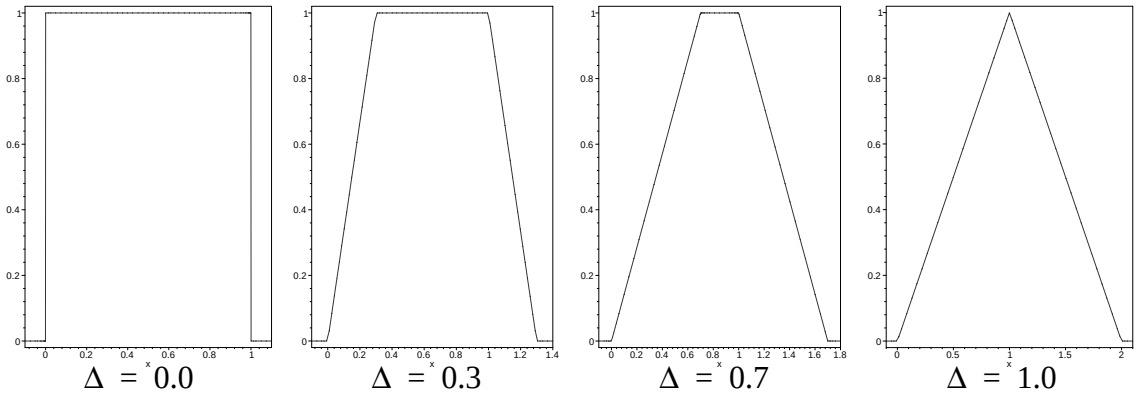


Figure 4.12 The BLaC scalar function for different values of Δ

This set of basis functions do not define a set of nested spaces, since the two scale relation is not satisfied. However the two scale relation can still be used to approximate the function at a different scale:

$$\phi^{\Delta}_{j,k}(x) \cong \tilde{\phi}^{\Delta}_{j,k}(x) = \alpha \cdot \phi^{\Delta}_{j+1,2k-1}(x) + \phi^{\Delta}_{j+1,2k}(x) + (1-\alpha) \cdot \phi^{\Delta}_{j+1,2k+1}(x) \quad \text{(EQ 4.76)}$$

where the value of α can be computed as the solution of

$$\frac{\partial}{\partial a}\left\| \phi^{\Delta}_{j,k}(x) - \tilde{\phi}^{\Delta}_{j,k}(x) \right\| = 0 \quad \text{(EQ 4.77)}$$

Eq. 4.77 has to be interpreted as a step that minimizes the non-orthogonality of $\phi^{\Delta}_{j,k}(x)$.

Figure 4.13 shows the approximate functions generated by the two scale relation:

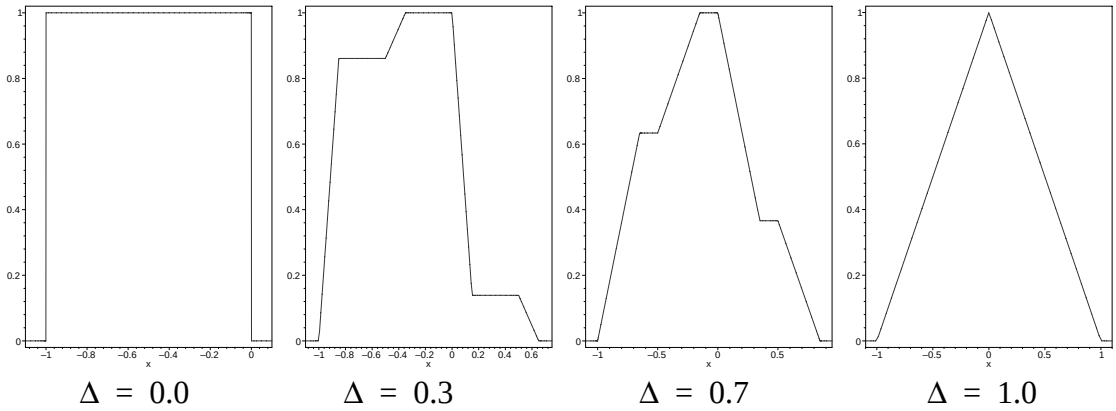


Figure 4.13 Function constructed with the two-scale relation for different values of Δ

Now that the scalar basis functions have been constructed, it is necessary to define and build the wavelet basis functions. The wavelet functions must satisfy the two-scale relation, which means that the wavelet bases at level j must be expressed as a linear combination of the scalar bases at level $j+1$, as shown in Eq. 4.78:

$$\psi_{j,k}^{\Delta}(x) = \sum_{l=-\infty}^{\infty} q_k \cdot \phi_{j+1,l}^{\Delta}(x) \qquad \text{(EQ 4.78)}$$

Since the size of the support of the wavelet basis functions is bounded by the size of the support of the linear spline wavelet basis function, it is possible to reduce this infinite sum to a sum with six terms:

$$\psi_{j,k}^{\Delta}(x) = \sum_{l=0}^{5} q_k \cdot \phi_{j+1,l}^{\Delta}(x) \qquad \text{(EQ 4.79)}$$

In order to construct a unique set of wavelet basis functions the following two constraints are imposed on the wavelet functions:

$$\langle \psi_{j,k}^{\Delta}(x), \tilde{\phi}_{j,l}^{\Delta}(x) \rangle = 0, \quad l = -\infty, \ldots, \infty \qquad \text{(EQ 4.80)}$$

$$\left\| \psi_{j,k}^{\Delta}(x) \right\| = 1 \qquad \text{(EQ 4.81)}$$

These two constraints allow us to determine the value of the coefficients q_i in Eq. 4.79 and thus evaluate the wavelet basis functions.

Some examples of wavelet basis functions for different values of Δ are presented in Figure 4.14:

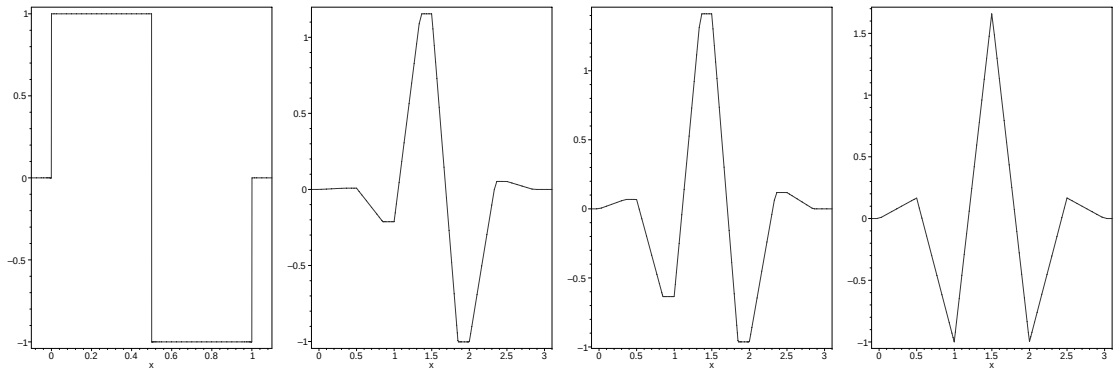


Figure 4.14 BLaC wavelet basis functions for different values of Δ

One final remark: the standard filter bank algorithm (see Section 4.1.3) needs to be extended in order to be used with the BLaC wavelets, since the spaces V_n are not nested. A single step of the filter bank algorithm is shown in Figure 4.15:

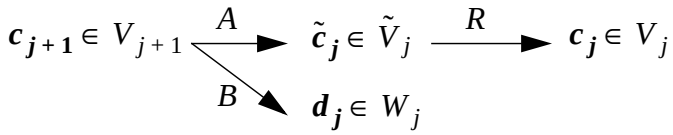


Figure 4.15 Filter bank algorithm for non-nested spaces

The input signal $\boldsymbol{c_{j+1}}$ is projected onto the space generated by $\tilde{\phi}^{\Delta}_{j,k}(x)$, the scalar basis function that is generated by the two scale relation, using the analysis operator A. The result $\tilde{\boldsymbol{c}}_{\boldsymbol{j}}$ is then projected onto the space generated by $\phi^{\Delta}_{j,k}(x)$, the correct BLaC basis function for the resolution j, using the projection operator R.

4.3.2 Multiresolution Analysis on Irregular Meshes

The idea of using non-nested spaces to represent functions can be generalized to meshes with arbitrary connectivity.

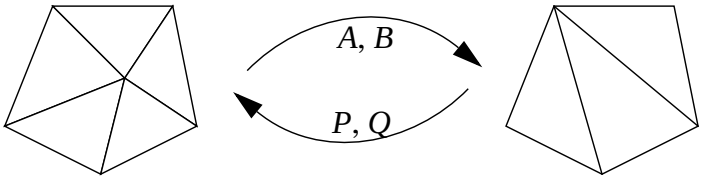
The construction of wavelets over arbitrary connectivity is complicated, since the scalar and wavelet basis functions not only depend on the geometric information, as in the case of tensor product wavelets and second generation wavelets, but also on the connectivity of the mesh. The idea of Bonneau is to avoid the construction of any basis function explicitly: instead he builds directly the analysis and synthesis matrices that are used in the filter-bank algorithm. In this way it is possible to construct compact *pseudo-wavelets* that are easy to implement, but are not as rigorous as standard wavelets.

The multiresolution analysis that Bonneau constructed has the following properties:

- The pseudo-wavelets can be used on meshes with arbitrary connectivity, but the macro-topology of the mesh is restricted: the mesh must be either parametrized over a plane defined as an height field or over a sphere. This is a mandatory requirement, since the algorithm needs a parametrization to build the analysis and synthesis matrices.
- The pseudo-wavelets do not encode the geometric information of the vertices that define the mesh; instead they code one scalar value per triangle. The topologic and geometric information has to be stored using conventional mesh-based algorithms.
- The construction of the multiresolution representation for meshes is based on a standard mesh decimation algorithm (see Section 7). In the original paper Bonneau used the vertex removal algorithm constructed by William Schroeder (see Section 7.1).

We saw in Section 4.1.3 that in order to construct a filter bank algorithm two analysis matrices A and B and and two synthesis matrices P and Q must be constructed.

In order to better describe the constructions of these matrices it is worthwhile to visualize the vertex removal operation which maps a mesh at level n to the mesh at level $n-1$:



a) Number of triangles: t_n
Level: n

b) Number of triangles: t_{n-1}
Level: $n-1$

Figure 4.16 The vertex removal operation
a) Region of the mesh at level *n*
b) Same region of the mesh at level *n-1* after the vertex removal operation

The matrix A is built using a simple strategy: the value of one triangle at a level $n-1$ is computed as the weighted average of the values of the triangles at level n. The weights are computed as the area of the intersection of two triangles in parameter space:

$$a_{i,j} = \left(\frac{\text{area}\ (T_i^{n-1} \cap T_j^n)}{\text{area}\ (T_i^{n-1})} \right) \qquad \text{(EQ 4.82)}$$

where T_i^{n-1} is the projection of the i-th triangle in the mesh at level $n-1$ in parameter space, and T_j^n is the projection of the j-th triangle in the mesh at level n in parameter space.

This means that the value of $a_{i,j}$ is different from zero only if the i-th triangle in the coarse mesh intersects the j-th triangle in the fine mesh. The matrix A has a size of $t_{n-1} \times t_n$.

The only requirement imposed on the matrix B is to store enough information to be able to reconstruct $\boldsymbol{c^n}$. This leaves a lot of freedom in the choice of B. An additional requirement imposed by Bonneau is that B must be orthogonal to the matrix A. This can be accomplished very easily by setting

$$B = \begin{bmatrix} I & X \end{bmatrix} \qquad \text{(EQ 4.83)}$$

where I is the identity matrix of size $(t_n - t_{n-1}) \times (t_n - t_{n-1})$, and X is an unknown matrix of size $(t_n - t_{n-1}) \times t_{n-1}$. The value of the sub-matrix X is computed as the solution of

$$\text{row}_i(B) \cdot \text{row}_j(A) = \delta_{i,j} \quad \text{(EQ 4.84)}$$

which corresponds to $t_n - t_{n-1}$ linear systems of equations with t_{n-1} unknowns each. The rows of the matrix B can also be orthogonalized using a Gram-Schmidt orthogonalization. The resulting matrix B has a size of $(t_n - t_{n-1}) \times t_{n-1}$.

Finally, the synthesis matrices P and Q are computed using Eq. 4.85:

$$(P \; Q) = \begin{pmatrix} A \\ B \end{pmatrix}^{-1} \quad \text{(EQ 4.85)}$$

4.4 Wavelets Over Manifolds

There is little literature on wavelets over manifolds. The main problem with this type of wavelets is that the bases depend on the connectivity of the underlying mesh, and it is still not clear how to construct proper wavelets in this setting. Wavelets also need a global parametrization, and the construction of parametrizations over meshes with arbitrary topology is still an open research topic.

Nonetheless there are some papers on the subject:

- One of the most interesting papers on this subject was written by Wolfang Dahmen in [87]. The paper is still very theoretical and does not give to much insight on how to build wavelets over manifolds.
- A very successful approach has been constructed by Lounsbery, who builds wavelets over subdivision surfaces. This strategy called Subdivision Wavelet Transform (SWT) has been developed by Lounsbery in [57], [23], [56], and it will be described in detail in Section 5.2. This representation, described in detail in Section 5.2, has been expanded by Eck et al. in [31] and by Guskov et al. in [46] to handle meshes with arbitrary connectivity.
- Andreas Dreger is working on the construction of B-Spline wavelets over quaternary subdivision meshes. Information on his work can be found in [27].

4.5 Evaluation

In this subsection we will evaluate the wavelet based representations described in Section 4 using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: wavelet based representations are defined in a rigorous mathematical framework. One of the consequences of this rigor is they cannot model surfaces defined in irregular settings.
  The three representations described in this section have different capabilities:

- *Tensor-product wavelet* by definition can only be used to model height fields defined over regular grids. Furthermore the size of the grid is usually restricted to a power of two plus a constant.

  The basic wavelet representation could be generalized to handle more complex surfaces, such as:

  1. *grids with irregular boundaries*: surfaces defined over regular grids with irregular boundaries can be encoded using wavelets. This can be accomplished by filling the missing values around the irregular boundary, until the boundary becomes regular.
     The choice of the missing values should minimize the detail values in the wavelet representation and therefore it depends on the scalar and wavelet basis functions.

  2. *structured grids*: store a (x, y, z) value per grid point instead of one single height value. Structured grids allow to model slightly more complex surfaces, such as salt domes.
     Structured grids can be modeled using the wavelet representation by computing three wavelet decompositions for the x, y, and z values. A change of the coordinate system via a *principal component analysis* (PCA) would probably result in a better representation.

  3. *Two-manifold surfaces*: there is no elegant solution to the problem of representing two-manifold surfaces with arbitrary connectivity with first generation wavelets.
     Two-manifold surfaces could be represented with tensor product wavelets by fitting B-Spline patches through the vertices that define the surface, and then compute a wavelet representation for each of the patches. However it is not clear how to keep cross-boundary continuity across neighboring patches.
     For more information on how to fit B-Spline patches to unstructured clouds of points or to polygonal surfaces see for example [32] and [49].

- *Second generation wavelets* are more general than standard tensor product wavelets as it is theoretically possible to construct basis functions for more general settings than $L^2(R)$.
  The bases that have been constructed until now usually handle simple grids with uniform or non-uniform sampling in u and v as well as two-manifold triangular meshes with subdivision connectivity.

- The *pseudo-wavelets* constructed by Bonneau, as we have seen in Section 4.3.2, are capable of modeling surfaces defined as height fields with arbitrary connectivity. His work could be extended to two-manifolds, using any of the local or global parametrization algorithms for 3D meshes, such as the hinge map used in [46] or the global parametrizations constructed in [51] and in [30].

  The most important drawback of this approach is that the multiresolution representation does not encode the geometric and topological information of a mesh, but merely a scalar value per triangle. The geometric and topologic information is stored using a classic mesh representation. In the original paper Bonneau used the

vertex removal scheme of W. Schroeder presented in [74].
Furthermore Bonneau never constructs the wavelet basis explicitly; instead he simply constructs the analysis and synthesis matrices that are then used in the filter bank algorithm. Consequently the properties of the basis functions are unknown.

- *Computation of surface-surface intersections*: the wavelet representation of a mesh does not contain the information required to compute an intersection. This means that the information must be either stored in a different structure or be computed when needed. There are two main techniques that can be used to compute intersections:
  - *Bounding boxes*: a hierarchical bounding box data structure is usually built as a tree, each node containing the size of the cube that bounds a portion of the surface. Leaf nodes in the tree contain cubes which bound small patches of the surface, for example a couple of triangles, the root node contains a cube that bounds the whole surface. This data structure allows to find triangles interested in an intersection in $O(n \cdot \log n)$ in average, a much faster alternative than a brute force test, which has a complexity of $O(n^2)$.
    This hierarchical structure cannot be integrated in the wavelet representation, but must be kept separately. Furthermore the information stored in this structure is not retrieved from the wavelet representation, but from the piecewise linear representation on the surface.
  - *Algebraic methods*: if the surface is represented using higher order functions it is possible to use complex methods to construct robust surface-surface intersection algorithms. One well known example is represented by the family of the algebraic methods. This family of algorithms can be applied to some of the representation generated by the tensor product and second generation wavelets.
    In [62] and in [50] S. Krishnan and D. Manocha presented an algorithm to compute the intersection of Bezier patches. This algorithm could be generalized to handle the surfaces stored in a wavelet representation that uses B-Spline basis functions. A survey on this technique can be found in [41].
- *Scalable representation*: the wavelet based representations presented in this section are all scalable, since they build a compact multi-resolution representation of a surface using a coarsification operator represented by the analysis matrices A and B.
  - The wavelet representation is compact: the wavelet based multiresolution representation of a surface has the same storage requirements than the surface itself, which means that the representation of a surface of size $n \times n$ also has a size of $n \times n$.
  - For *first generation wavelets* if the basis functions are orthogonal, the most important information can be extracted easily from the wavelet representation, since there is a direct correlation between the importance of a detail value $d_{i,j}$ and its absolute value.

    The basis functions generated using the lifting step in the *second generation wavelets* are in general only bi-orthogonal. As a consequence it is much more difficult to find the optimal subset of detail values that can be removed to obtain an best

approximation of the surface. It is nonetheless possible to remove details with small absolute value and obtain good approximations.

The basis functions used by Bonneau in his *pseudo-wavelets* are not known, so it is not clear how to find the optimal set of detail values that must be removed to obtain the best approximation of the surface. However it is possible to leave out details with small absolute value an obtain good approximations.

- The algorithms used to build the wavelet representation have a very low complexity: it is possible to construct *tensor product wavelets* based on endpoint interpolating cubic B-Spline in $O(n)$ time and storage requirements. As shown in Section 4.2 the decomposition and reconstruction of *second generation wavelets* can be computed in linear time using the fast wavelet transform. The *pseudo-wavelets* presented in Section 4.3 can also be constructed in linear time.

- *Modeling of non-manifold singularities, tears and cracks*: *first* and *second generation wavelets* are not capable to model non-manifold surfaces, since they cannot be represented in terms of the regular spaces these frameworks work on, such as grids or quaternary subdivision surfaces. Tears and cracks can be embedded as piecewise linear curves in the piecewise linear polygonal mesh.

  The pseudo-wavelets constructed by Bonneau are based on an underlying surface representation. In his paper Bonneau used the Schroeder's algorithm to simplify the mesh. Since this simplification algorithm can handle non-manifolds (see Section 7.1 for more details) it is plausible to extend the pseudo-wavelets over non-manifolds. The main issue that needs to be solved is to find a good parametrization to compute Eq. 4.82.

- *Error modeling*: the wavelet theory is especially powerful at modeling the error of a surface; some theoretic results can be used to construct error models that evaluate the error introduced by an adaptive representation of surfaces.

  The error associated with bases generated in the classic *tensor product* setting is well understood, and good error estimates can be built, particularly if the basis functions are orthogonal.

  The error associated with bases constructed with the *lifting scheme* must be constructed explicitly. If the basis can be constructed using standard first generation techniques, then the error is well understood, otherwise a suitable error model must be constructed.

  The error associated with the pseudo-wavelets is not well understood since, as already mentioned, the basis functions are never constructed explicitly: instead the algorithm builds the analysis and synthesis matrix from the 3D mesh. As a consequence this representation has not been studied using tools from approximation theory.

  One point that should be investigated further is how the error should be defined: the norm used to compute the error during a flow simulation is probably different from the norm used to compute the error during an interactive visualization of a surface. It would be of great interest to develop norms that could capture these different types of error.

- *Smoothness of the surface*: if the surface is being represented with either *first* or *second generation wavelets* the smoothness of the surface depends on the choice of the basis functions. If the surface is encoded using the Haar basis function described in

Section 4.1.6, then the surface is defined as piecewise constant; using a cubic B-Spline basis function will result in a smooth C^2 surface.

An analysis of the *pseudo-wavelets* generated by Bonneau suggests that the representation is not smooth, and that the surface is represented as piecewise constant. The lack of explicit basis functions proves to be a problem that makes a good understanding of the representation difficult.

- *Multiresolution editing*: edit operations at different levels of resolution can be applied on surfaces represented using *first* and *second wavelet framework*, because:
  - surfaces are represented as (piecewise defined) functions. This means that an edit operation changes the shape of a function and not the position of a single vertex in the mesh, and this allows to compute natural changes.
  - in the multiresolution setting, the scalar values $\boldsymbol{c^n}$ that define the surface at level n and the detail values $\boldsymbol{d^n}$ that are used to reconstruct the scalar values $\boldsymbol{c^{n+1}}$ are orthogonal. This means that an edit at the level n need not be propagated to the levels $j > n$, but only to the levels $j < n$.

  The representation proposed by Bonneau generates a multiresolution analysis, and the information needed to transform a coarse mesh at level n to a finer mesh at level $n-1$ is coded with differences and not with absolute values. This means that a multiresolution editing tool could be built easily. On the other hand the lack of explicit basis functions makes it difficult to understand how the surface would change with an edit operation, since the basis functions have a direct impact on the edits.
- *Surface fitting*: for *first* and *second generation wavelets* the fitting process heavily depends on the basis functions chosen for the representation. Some basis functions are well understood, and it is possible to construct fitting schemes very easily; other basis functions can result in more elaborate schemes.
  Example: fitting a height field defined over a regular grid through a cloud of points using a cubic B-Spline basis function is a straightforward operation: Forsey [34] uses a similar strategy to fit an H-Spline surface.

  Fitting is currently not supported by the *pseudo-wavelets*, since the representation does not work on either the topological or geometric information. The fitting process would probably need to be a pre-processing step, and it would not automatically generate the multiresolution representation described in this section.
- *Support of local high variation in the curvature of the surface*: the representation generated using *first* or *second generation techniques* can model high variation in the curvature only if the basis functions chosen are not smooth. If a surface is represented using smooth basis functions, for example by using uniform cubic B-Spline basis functions, then it is not possible to model discontinuities in the curvature; if a surface is represented using linear or constant basis functions, then high variation can be modeled easily, since no continuity in the curvature is neither required nor guaranteed.

  The representation of Bonneau based on the pseudo-wavelets supports high variation in the curvature well, since the underlying surface representation based on Schoeder's algorithm can support this feature. High variation can be maintained by not removing

the features in the mesh that generates the high curvature. A more detailed description of the algorithm developed by W. Schroeder can be found in Section 7.1.

- *Changes of the surface over time*: in the classic *tensor product setting* time can be modeled elegantly by adding one more dimension to the wavelet representation and making use of the tensor product ansatz.

  Bases generated using the *lifting step* can model time as easily as first generation wavelets only if they are defined using tensor product. If the basis functions are more complicated, the ability of modeling time depends on the basis and on the domain this basis is working on. In the worst scenario it might be necessary to keep different copies of the representation for every point in time t_i.

  The complexity of topological changes of the surface over time determines how easily the *pseudo wavelet representation* can handle time:

  1. If the changes of the surface do not alter its topology but only the scalar values associated with each triangle, then these changes can be modeled very easily by storing a function $f(t)$ per triangle, which depend on the time t, instead of a single scalar value.
  2. If the changes affect the connectivity, but the macrotopology of the mesh does not change, it is possible to compute a remeshing of the surfaces at each point in time t_i in order to generate new surfaces with the same connectivity. Once all the surfaces have the same connectivity approach 1. can be used to represent changes over time.
  3. If the topological type of the mesh changes, the only solution is to keep different representations of the surfaces at each point in time t_i.

5 Subdivision Surfaces

5.1 Classic Subdivision Schemes

An alternative to wavelet representations are subdivision schemes. Wavelets use sets of basis functions to decompose a signal and create a multiresolution representation; subdivision surfaces do not work explicitly with basis functions, but they use fixed schemes to refine surfaces. The limit surface generated by an infinite number of refinement operations using a subdivision rule on an input mesh is provably smooth.

A subdivision scheme can be described with:

- a *topological component*: every scheme changes the microtopology of surfaces by adding and/or removing vertices and by changing their connectivity. Subdivision schemes can further be categorized as
  - *primal*: primal schemes subdivide each face of the input mesh in a number of subfaces. An example of a primal scheme is the *quaternary subdivision* scheme, where a triangle face is subdivided into four faces. To accomplish this three new vertices are introduced on the edges that define the original triangle.
  - *dual*: dual schemes not only introduce new vertices in the mesh, but they also remove old vertices and change the connectivity of the input mesh. This implies that a new face can span more than one face of the original mesh.
- a *geometric component*: the change in the position of the vertices can be interpreted as a low-pass filtering of the mesh, and as a result the surface is smoothed. Subdivisions can be categorized as
  - *interpolating*: the position of the vertices of the original mesh does not change, the position of the new vertices can be decided freely.
  - *non-interpolating*: the position of the old and the new vertices can be changed as needed.

Subdivision schemes are not a new surface representation: the first papers on this topic were published in 1978, but new papers on the subject are still being published, since subdivision is a very powerful operator particularly useful in animation. During this long period many different schemes have been proposed, the most important ones being:

- Doo-Sabin (1978): a non-interpolating dual scheme [26] and [25].
- Catmull-Clark (1978): a non-interpolating primal scheme [9].
- Loop (1987): a non-interpolating primal scheme [55].
- Butterfly (1990): an interpolating primal scheme [29].

In the next subsections we are going to describe these four schemes:

- We will present the Doo-Sabin scheme in depth: we will show how the scheme is constructed in a regular setting, and how to extend it to meshes with irregular topology.

Then we will show how to generalize it to non-uniform subdivision in order to better model non-smooth regions of the surface.

- The remaining schemes will not be treated in full detail: for each scheme we will only give the topological changes introduced by a refinement step as well as the masks used to compute the new vertex positions.

For additional information on the scheme that will be introduced in this section refer to the original papers.

5.1.1 Doo-Sabin Scheme

The scheme developed by Donald Doo and Malcom Sabin is based on a bi-quadratic uniform B-Spline refinement: the limit surface of this subdivision scheme is a bi-quadratic B-Spline patch; it generates a G^1 surface, which is a C^1 continuous surface everywhere except at a finite number of points, called *extraordinary points*.

In the following we will present the classic Doo-Sabin scheme for the one dimensional case, for the two-dimensional case over a regular grid, and for the two-dimensional case for surfaces with arbitrary connectivity. More detailed information on this standard scheme can be found in [26] and in [25].

Once the basic subdivision scheme has been defined we will present an extension that generates in the limit non-uniform recursive quadratic B-Spline surfaces. This new scheme has been constructed by T. Sederberg et al. in [75].

5.1.1.1 One dimensional uniform subdivision

The one dimensional scheme is the easiest scheme to construct, since the connectivity of the curve is trivial. The tensor product ansatz will allow us to generalize the results obtained in this section to the regular two-dimensional setting.

The input of the algorithm is a piecewise linear curve. The vertices $\boldsymbol{d_i}$ that specify the curve are considered as the control points of the quadratic B-Spline surface defined as

$$S(u) = \sum_{i=1}^{n} \boldsymbol{d_i} \cdot N_i^2(u) \qquad \text{(EQ 5.1)}$$

where $N_i^2(u)$ is a quadratic B-Spline that can be constructed using Eq. 4.31 and Eq. 4.32, and u is the parameter value. One important property of the Doo-Sabin subdivision is that it works with uniform quadratic B-Splines, which means that

$$\Delta u_i = u_i - u_{i-1} \qquad \text{(EQ 5.2)}$$

is a constant. In Section 5.1.1.4 through Section 5.1.1.6 this method will be generalized to a non-uniform quadratic B-Spline, where the size of the intervals in the parameter space will vary.

The curve generated by Eq. 5.1 can be evaluated in three different ways:

1. The value of $S(u)$ in Eq. 5.1 can be evaluated directly by computing the values of the B-Spline bases $N_i^2(u)$.

2. The value of $S(u)$ can also be computed using the de Boor algorithm, which uses linear interpolation to evaluate Eq. 5.1:

$$\boldsymbol{d}_i^k = (1 - a_i^k) \cdot \boldsymbol{d}_{i-1}^{k-1} + a_i^k \cdot \boldsymbol{d}_i^{k-1} \qquad \text{(EQ 5.3)}$$

   where

$$a_i^k = \frac{u - u_i}{u_{i+3-k} - u_i} \qquad \text{(EQ 5.4)}$$

   $\boldsymbol{d}_i^0 = \boldsymbol{d}_i$, and $\boldsymbol{d}_i^2 = S(u)$.

3. The third possible strategy used to compute $S(u)$ consists in inserting new knots at the midpoint of each knot interval, as shown in Figure 5.1.

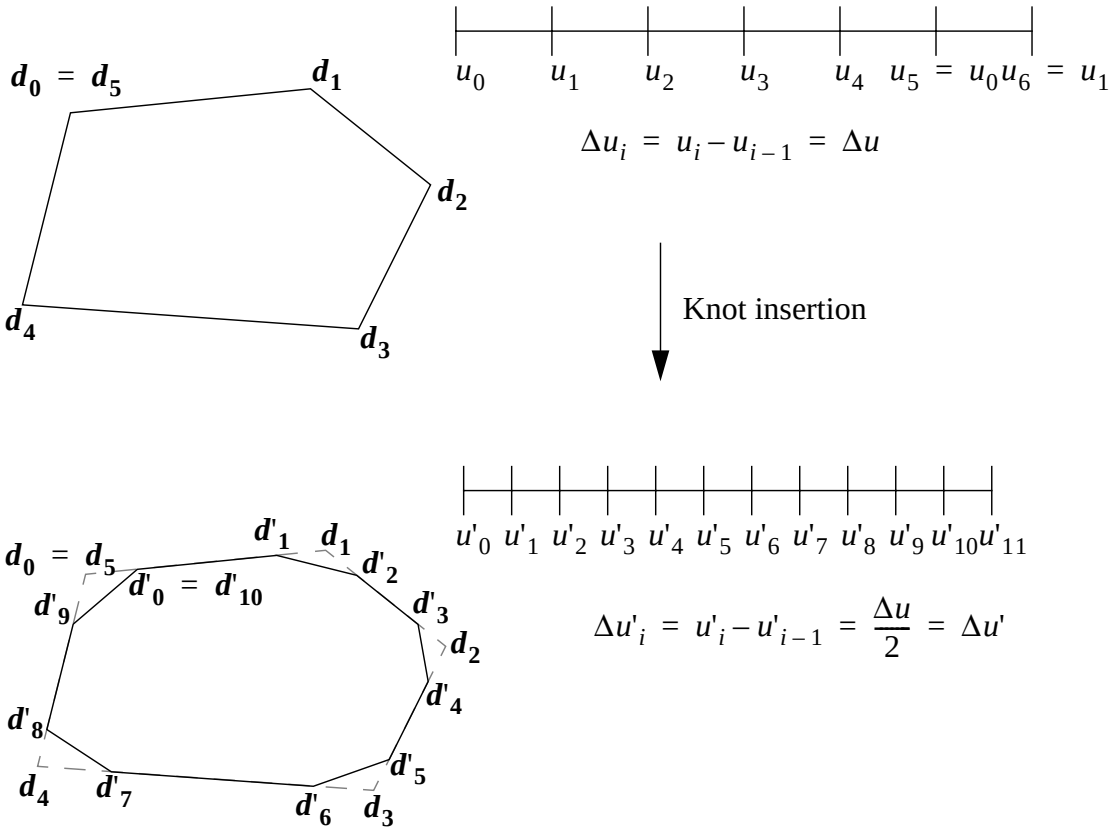


Figure 5.1 Knot insertion for the uniform quadratic B-Spline

After an infinite number of knot insertion operations at midpoint of the knot intervals the set of control points $\boldsymbol{d'_i}$ will converge to the curve $S(u)$.

The position of the new control points $\boldsymbol{d'_i}$ is computed via the knot insertion algorithm and is based on the de Boor algorithm:

$$\boldsymbol{d'_{2i-1}} = \frac{1}{4} \cdot \boldsymbol{d_{i-1}} + \frac{3}{4} \cdot \boldsymbol{d_i} \qquad \text{(EQ 5.5)}$$

$$\boldsymbol{d'_{2i}} = \frac{3}{4} \cdot \boldsymbol{d_i} + \frac{1}{4} \cdot \boldsymbol{d_{i+1}} \qquad \text{(EQ 5.6)}$$

where the weights of $\frac{1}{4}$ and $\frac{3}{4}$ have been computed assuming that the knots have a uniform distance.

The knot insertion operation described by Eq. 5.6 and Eq. 5.6 is presented in Figure 5.2.

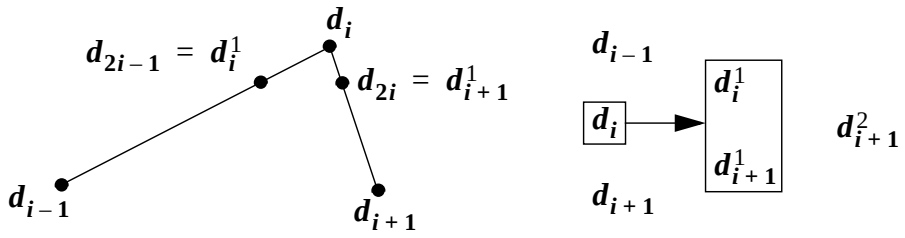


Figure 5.2 Computation of the new control points for the uniform quadratic B-Spline

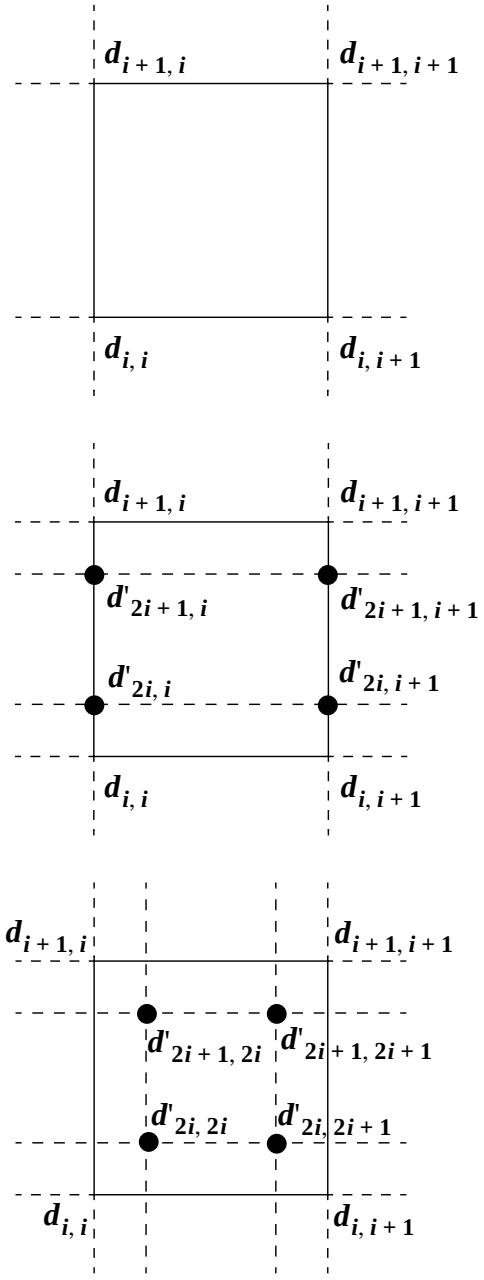
5.1.1.2 Two dimensional uniform subdivision over regular grids

The first extension to the one dimensional subdivision is a two dimensional scheme that makes use of the tensor-product ansatz. Topologically, the mesh of control points is a quadrilateral.

As we have seen in Section 5.1.1.1 a subdivision scheme computes the value of

$$S(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{m} \boldsymbol{d_{i,j}} \cdot N_i^2(u) \cdot N_j^2(v) \qquad \text{(EQ 5.7)}$$

by doubling the number of control points in the control matrix. The evaluation of Eq. 5.7 in two dimensions is very similar: Eq. 5.6 and Eq. 5.6 are first applied in the u direction, then in the v direction. This approach is called *separable construction*:



$$d'_{2i+1,i} = \frac{1}{4}d_{i,i} + \frac{3}{4}d_{i+1,i}$$

$$d'_{2i,i} = \frac{3}{4}d_{i,i} + \frac{1}{4}d_{i+1,i}$$

$$d'_{2i+1,i+1} = \frac{1}{4}d_{i,i+1} + \frac{3}{4}d_{i+1,i+1}$$

$$d'_{2i,i+1} = \frac{3}{4}d_{i,i+1} + \frac{1}{4}d_{i+1,i+1}$$

$$d'_{2i+1,2i} = \frac{3}{4}d'_{2i+1,i} + \frac{1}{4}d'_{2i+1,i+1}$$

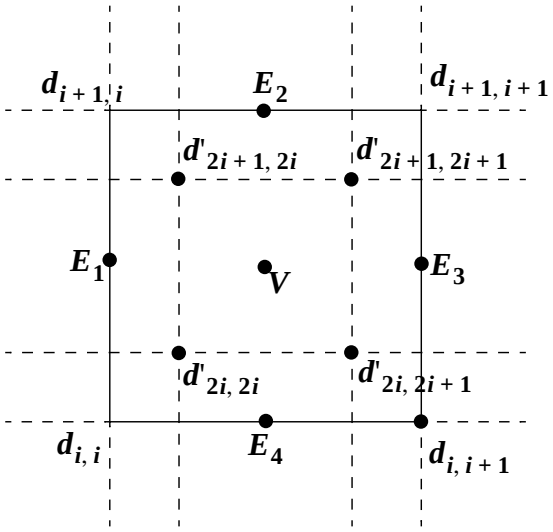
$$d'_{2i,2i} = \frac{3}{4}d'_{2i,i} + \frac{1}{4}d'_{2i,i+1}$$

$$d'_{2i+1,2i+1} = \frac{1}{4}d'_{2i+1,i} + \frac{3}{4}d'_{2i+1,i+1}$$

$$d'_{2i,2i+1} = \frac{1}{4}d'_{2i,i} + \frac{3}{4}d'_{2i,i+1}$$

Figure 5.3 Knot insertion in two dimensions for the uniform Doo-Sabin subdivision

The results presented in Figure 5.3 can be re-written in a form similar to the well-known Doo-Sabin subdivision rule, as shown in Figure 5.4:



$$V = \frac{d_{i,i} + d_{i+1,i} + d_{i+1,i+1} + d_{i,i+1}}{4}$$

$$E_1 = \frac{d_{i,i} + d_{i+1,i}}{2}$$

$$E_2 = \frac{d_{i+1,i} + d_{i+1,i+1}}{2}$$

$$E_3 = \frac{d_{i+1,i+1} + d_{i,i+1}}{2}$$

$$E_4 = \frac{d_{i,i+1} + d_{i,i}}{2}$$

$$d'_{2i,2i} = \frac{d_{i,i} + V + E_4 + E_1}{4}$$

$$d'_{2i+1,2i} = \frac{d_{i+1,i} + V + E_1 + E_2}{4}$$

$$d'_{2i+1,2i+1} = \frac{d_{i+1,i+1} + V + E_2 + E_3}{4}$$

$$d'_{2i,2i+1} = \frac{d_{i,i+1} + V + E_3 + E_4}{4}$$

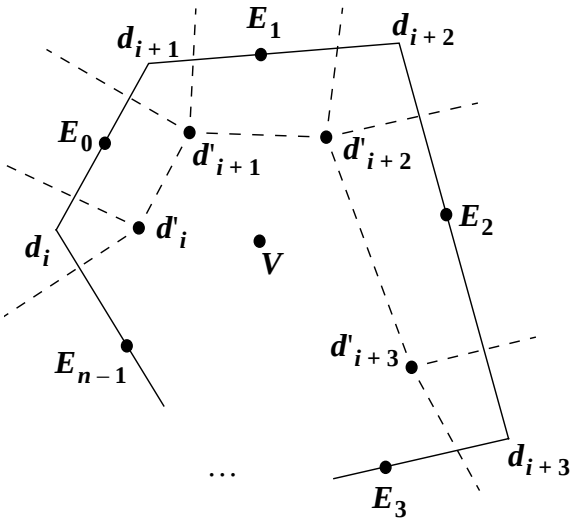
Figure 5.4 Uniform quadratic subdivision for regular grids

5.1.1.3 Two dimensional uniform subdivision for meshes with arbitrary topology

The Doo-Sabin subdivision can also be applied to meshes of control points that have arbitrary connectivity. The limit surface in this case is also a bi-quadratic B-Spline surface, but a finite number of extraordinary points on this surface will not be C^1 continuous. These extraordinary points correspond to the centers of all the non-quadrilateral faces generated after one subdivision step.

Meshes with arbitrary connectivity do not possess a natural global parametrization, so it is not possible to apply the de Boor algorithm in this setting to compute the position of the new vertices.

Although the connectivity of an input mesh is not restricted, a slightly modified version of the subdivision rule presented in Figure 5.4 can be used. The resulting scheme is shown in Figure 5.5.



$$V = \sum_{j=i}^{i+(n-1)} d_j$$

$$E_j = \frac{d_{i+j} + d_{i+(j+1) \bmod n}}{2}$$

$$d'_{i+j} = \frac{d_{i+j} + E_{(j-1) \bmod n} + E_j + V}{4}$$

Figure 5.5 Uniform quadratic subdivision for meshes with arbitrary connectivity

Figure 5.6 shows an example of a Doo-Sabin subdivision applied on a simple two-manifold surface with no boundaries:

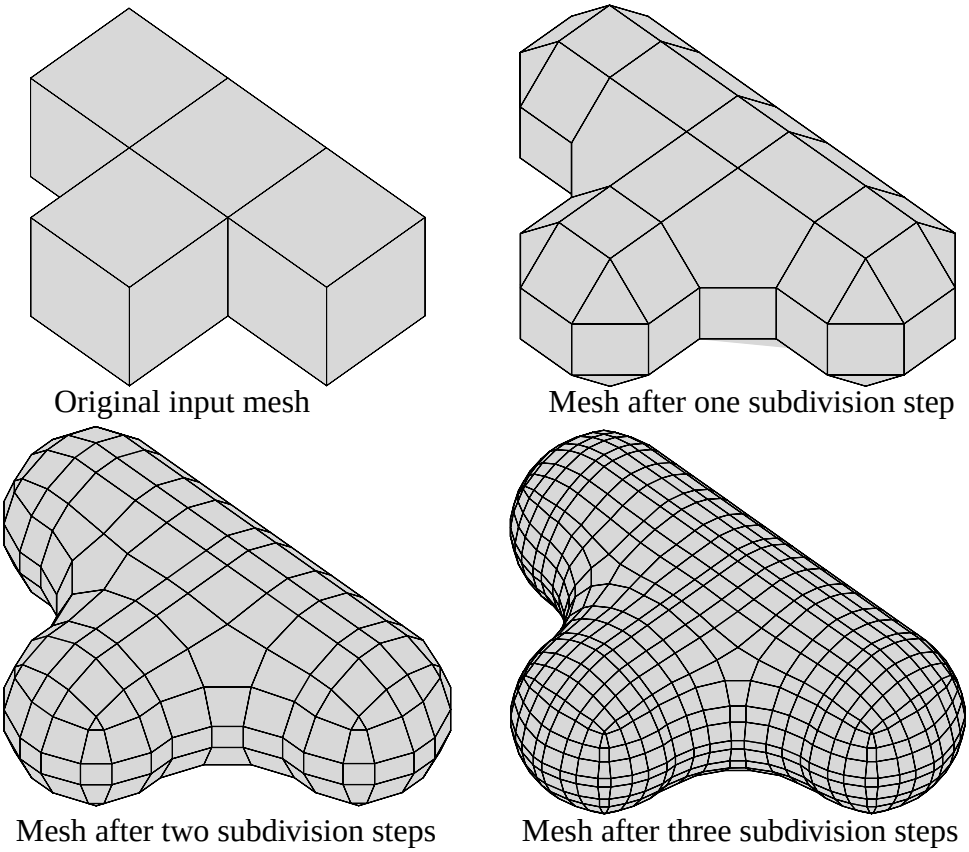


Figure 5.6 Example of the Doo-Sabin subdivision

5.1.1.4 One dimensional non-uniform subdivision

We now consider an extension of the classic Doo-Sabin scheme based on non-uniform quadratic B-splines. The basic algorithm is the same as the algorithm used in Section 5.1.1.1, but the resulting equations for the new control points $\boldsymbol{d'_{2i}}$ and $\boldsymbol{d'_{2i+1}}$ will differ from Eq. 5.6 and Eq. 5.6 in that they will consider the non-uniform knot distances

$$\Delta u_i = u_i - u_{i-1} \qquad \text{(EQ 5.8)}$$

The refinement step is computed via knot insertion at midpoint of each knot interval, in the same way it is computed for the uniform Doo-Sabin scheme. The refinement continues until the control polygon converges to the limit surface. A single knot insertion step for the non-uniform Doo-Sabin subdivision is given in Figure 5.7:

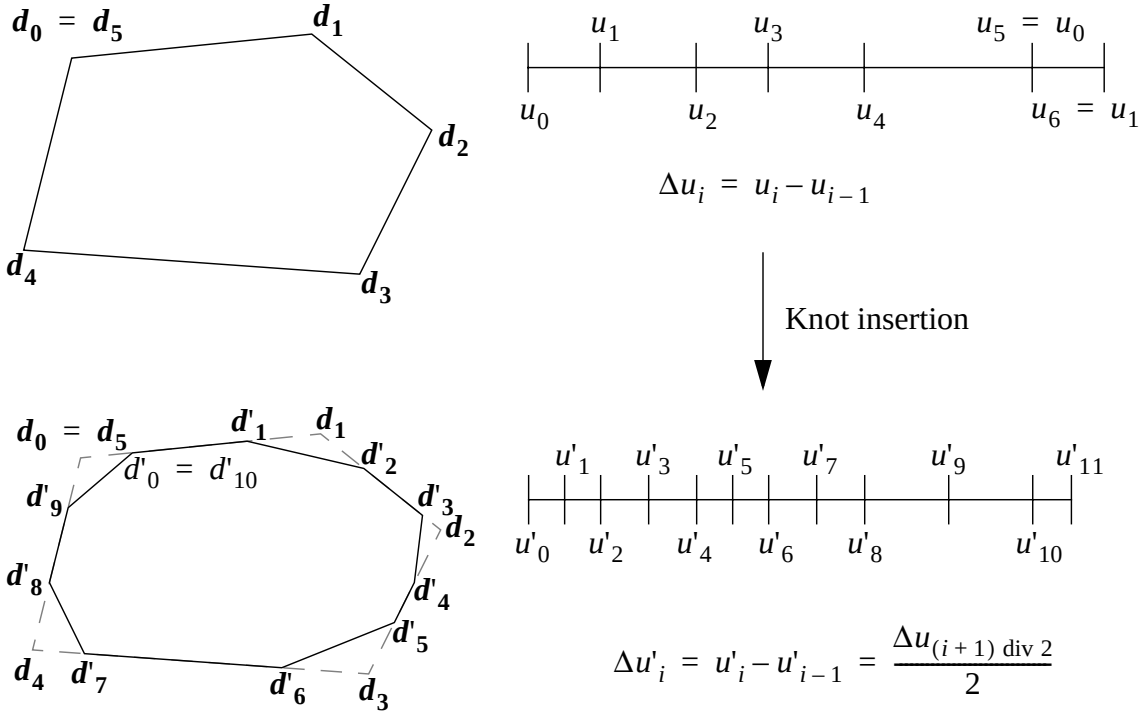
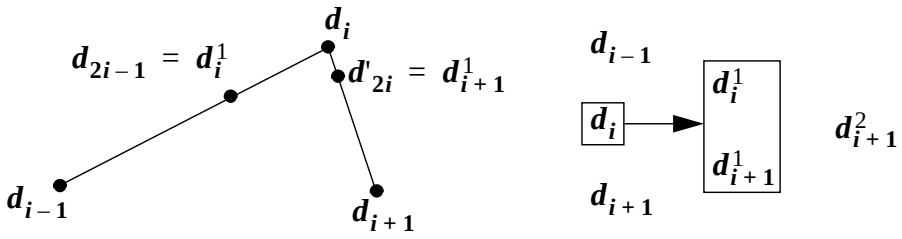


Figure 5.7 Knot insertion for the non-uniform quadratic B-Spline

The de Boor algorithm presented in Eq. 5.3 and Eq. 5.4 can also be used for the non-uniform case; Figure 5.8 shows how it is used to compute the position of the new control points:



$$d'_{2i-1} = d_i^1 = \frac{\left(\frac{1}{2}\Delta u_{i+2}\right)d_{i-1} + \left(\Delta u_{i+1} + \frac{1}{2}\Delta u_{i+2}\right)d_i}{\Delta u_{i+1} + \Delta u_{i+2}}$$

$$d'_{2i} = d_{i+1}^1 = \frac{\left(\frac{1}{2}\Delta u_{i+2} + \Delta u_{i+3}\right)d_i + \left(\frac{1}{2}\Delta u_{i+2}\right)d_{i+1}}{\Delta u_{i+2} + \Delta u_{i+3}}$$

Figure 5.8 Computation of the new control points for the non-uniform quadratic B-Spline

It is easy to check that by setting $\Delta u_{i+1} = \Delta u_{i+2} = \Delta u_{i+3} = \Delta u$, basically setting the knot intervals at uniform distances, the equations computed in Figure 5.8 reduce to Eq. 5.5 and Eq. 5.6 that were derived in Section 5.1.1.1.

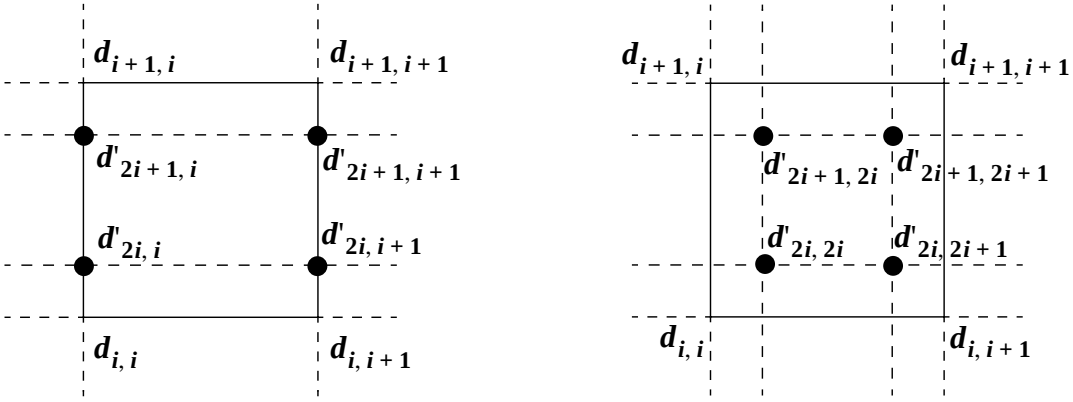
5.1.1.5 Two dimensional non-uniform subdivision over regular grids

The results described in Section 5.1.1.4 can be extended to regular two dimensional surfaces using the tensor product ansatz in the same way it was used in Section 5.1.1.2.

As in the uniform case the goal is to compute

$$S(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{m} d_{i,j} \cdot N_i^2(u) \cdot N_j^2(v) \qquad \text{(EQ 5.9)}$$

This is accomplished by first inserting new knots at the midpoints of the knot intervals of the knot vector u and then inserting new knots at the midpoints of the knot intervals of the knot vector v, as shown in Figure 5.9.



$$\boldsymbol{d}'_{2i,i} = \frac{\left(\frac{1}{2}\Delta u_{i+2} + \Delta u_{i+3}\right)\boldsymbol{d}_{i,i} + \left(\frac{1}{2}\Delta u_{i+2}\right)\boldsymbol{d}_{i+1,i}}{\Delta u_{i+2} + \Delta u_{i+3}}$$

$$\boldsymbol{d}'_{2i+1,i} = \frac{\left(\frac{1}{2}\Delta u_{i+3}\right)\boldsymbol{d}_{i,i} + \left(\Delta u_{i+2} + \frac{1}{2}\Delta u_{i+3}\right)\boldsymbol{d}_{i+1,i}}{\Delta u_{i+2} + \Delta u_{i+3}}$$

$$\boldsymbol{d}'_{2i,i+1} = \frac{\left(\frac{1}{2}\Delta u_{i+2} + \Delta u_{i+3}\right)\boldsymbol{d}_{i,i+1} + \left(\frac{1}{2}\Delta u_{i+2}\right)\boldsymbol{d}_{i+1,i+1}}{\Delta u_{i+2} + \Delta u_{i+3}}$$

$$\boldsymbol{d}'_{2i+1,i+1} = \frac{\left(\frac{1}{2}\Delta u_{i+3}\right)\boldsymbol{d}_{i,i+1} + \left(\Delta u_{i+2} + \frac{1}{2}\Delta u_{i+3}\right)\boldsymbol{d}_{i+1,i+1}}{\Delta u_{i+2} + \Delta u_{i+3}}$$

$$\boldsymbol{d}'_{2i,2i} = \frac{\left(\frac{1}{2}\Delta v_{i+2} + \Delta v_{i+3}\right)\boldsymbol{d}'_{2i,i} + \left(\frac{1}{2}\Delta v_{i+2}\right)\boldsymbol{d}'_{2i,i+1}}{\Delta v_{i+2} + \Delta v_{i+3}}$$

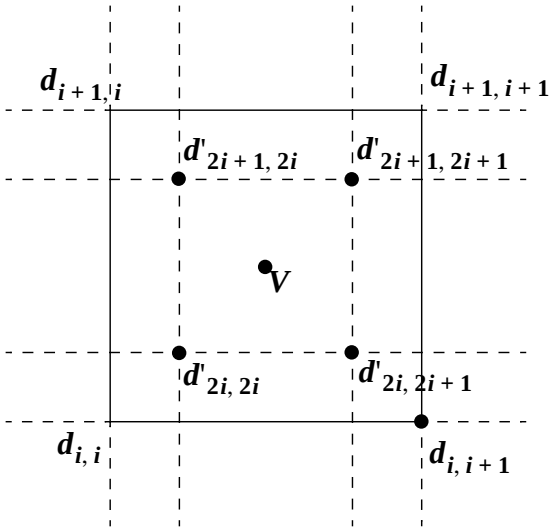
$$\boldsymbol{d}'_{2i,2i+1} = \frac{\left(\frac{1}{2}\Delta v_{i+3}\right)\boldsymbol{d}'_{2i,i} + \left(\Delta v_{i+2} + \frac{1}{2}\Delta v_{i+3}\right)\boldsymbol{d}'_{2i,i+1}}{\Delta v_{i+2} + \Delta v_{i+3}}$$

$$\boldsymbol{d}'_{2i+1,2i} = \frac{\left(\frac{1}{2}\Delta v_{i+2} + \Delta v_{i+3}\right)\boldsymbol{d}'_{2i+1,i} + \left(\frac{1}{2}\Delta v_{i+2}\right)\boldsymbol{d}'_{2i+1,i+1}}{\Delta v_{i+2} + \Delta v_{i+3}}$$

$$\boldsymbol{d}'_{2i+1,2i+1} = \frac{\left(\frac{1}{2}\Delta v_{i+3}\right)\boldsymbol{d}'_{2i+1,i} + \left(\Delta v_{i+2} + \frac{1}{2}\Delta v_{i+3}\right)\boldsymbol{d}'_{2i+1,i+1}}{\Delta v_{i+2} + \Delta v_{i+3}}$$

Figure 5.9 Knot insertion in two dimensions for the non-uniform Doo-Sabin subdivision

Also for the non-uniform subdivision it is possible to rewrite the results presented in Figure 5.9 in a form similar to the classic Doo-Sabin subdivision rule. This is done in Figure 5.10.



$$\boldsymbol{V} = \frac{\boldsymbol{d}_{i,i} + \boldsymbol{d}_{i+1,i} + \boldsymbol{d}_{i+1,i+1} + \boldsymbol{d}_{i,i+1}}{(\Delta u_{i+2} + \Delta u_{i+3}) \cdot (\Delta v_{i+2} + \Delta v_{i+3})}$$

$$\boldsymbol{d}'_{2i,2i} = \frac{\boldsymbol{d}_{i,i} + \boldsymbol{V}}{2} + \frac{\Delta u_{i+2}\Delta v_{i+2} \cdot (\boldsymbol{d}_{i+1,i} + \boldsymbol{d}_{i,i+1} - \boldsymbol{d}_{i,i} - \boldsymbol{d}_{i+1,i+1})}{4 \cdot (\Delta u_{i+2} + \Delta u_{i+3}) \cdot (\Delta v_{i+2} + \Delta v_{i+3})}$$

$$\boldsymbol{d}'_{2i,2i+1} = \frac{\boldsymbol{d}_{i,i+1} + \boldsymbol{V}}{2} + \frac{\Delta u_{i+2}\Delta v_{i+3} \cdot (\boldsymbol{d}_{i,i} + \boldsymbol{d}_{i+1,i+1} - \boldsymbol{d}_{i,i+1} - \boldsymbol{d}_{i+1,i})}{4 \cdot (\Delta u_{i+2} + \Delta u_{i+3}) \cdot (\Delta v_{i+2} + \Delta v_{i+3})}$$

$$\boldsymbol{d}'_{2i+1,2i} = \frac{\boldsymbol{d}_{i+1,i} + \boldsymbol{V}}{2} + \frac{\Delta u_{i+3}\Delta v_{i+2} \cdot (\boldsymbol{d}_{i,i} + \boldsymbol{d}_{i+1,i+1} - \boldsymbol{d}_{i,i+1} - \boldsymbol{d}_{i+1,i})}{4 \cdot (\Delta u_{i+2} + \Delta u_{i+3}) \cdot (\Delta v_{i+2} + \Delta v_{i+3})}$$

$$\boldsymbol{d}'_{2i+1,2i} = \frac{\boldsymbol{d}_{i+1,i+1} + \boldsymbol{V}}{2} + \frac{\Delta u_{i+3}\Delta v_{i+3} \cdot (\boldsymbol{d}_{i+1,i} + \boldsymbol{d}_{i,i+1} - \boldsymbol{d}_{i,i} - \boldsymbol{d}_{i+1,i+1})}{4 \cdot (\Delta u_{i+2} + \Delta u_{i+3}) \cdot (\Delta v_{i+2} + \Delta v_{i+3})}$$

Figure 5.10 Non-uniform Doo-Sabin subdivision for regular grids

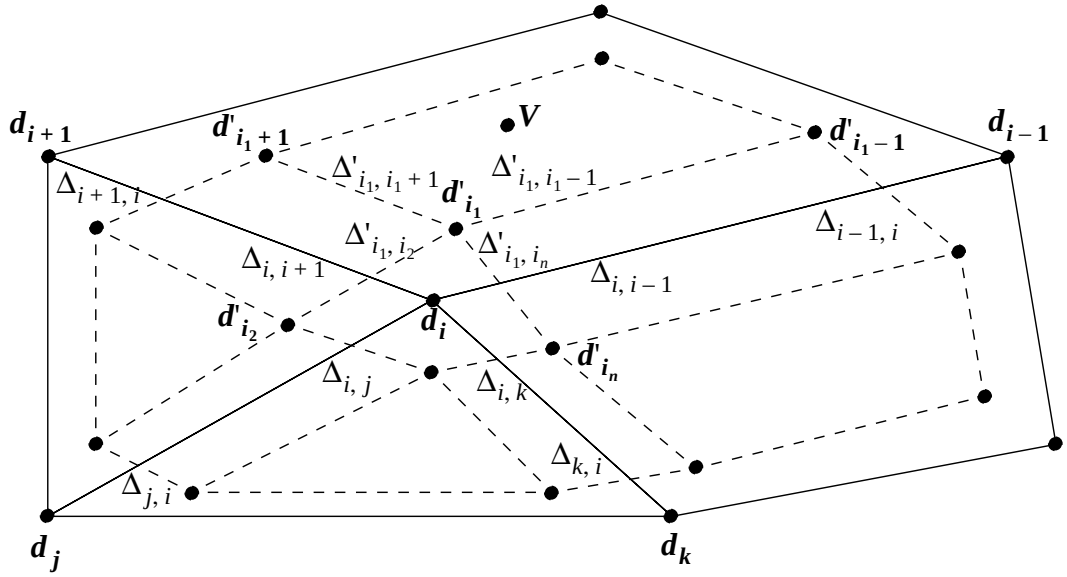
5.1.1.6 Two dimensional non-uniform subdivision for meshes with arbitrary connectivity

The results presented in Section 5.1.1.5 can be generalized to handle meshes with arbitrary topology.

In order to describe how to compute the refinement of the control points in this irregular setting it is necessary to introduce new parameters that will be used to describe the knot intervals. Two-manifold meshes with arbitrary topology do not possess a natural global parametrization. Since there is no u and v knot vector anymore, it is not convenient to denote a knot interval as Δu_i or Δv_i. In the remainder of this section a knot interval

between two vertices $\boldsymbol{d_i}$ and $\boldsymbol{d_j}$ will be called $\Delta_{i,\,j}$.

The refinement operator for the non-uniform quadratic subdivision for two-manifolds surfaces is shown in Figure 5.11.



I) Compute the position of the new control vertices

$$\boldsymbol{d'_{i_1}} = \frac{\boldsymbol{d_i} + \boldsymbol{V}}{2} + \boldsymbol{C}$$

$$\boldsymbol{V} = \frac{\sum_{k=1}^{n} \Delta_{k-1,\,k} \cdot \Delta_{k+1,\,k} \cdot \boldsymbol{d_k}}{\sum_{k=1}^{n} \Delta_{k-1,\,k} \cdot \Delta_{k+1,\,k}}$$

$$\boldsymbol{C} = (\Delta_{i+1,\,i+2}\Delta_{i+3,\,i+2} + \Delta_{i-1,\,i-2}\Delta_{i-3,\,i-2}) \cdot \frac{-n\boldsymbol{d_i} + \sum_{j=1}^{n} \left(1 + 2\cos\left(\frac{2\pi|i-j|}{n}\right)\right)\boldsymbol{d_j}}{8\sum_{k=1}^{n} \Delta_{k-1,\,k}\Delta_{k+1,\,k}}$$

n is the number of sides of the face

II) Compute the new knot intervals

$$\Delta'_{i_1,\,i_1+1} = \frac{\Delta_{i,\,i+1}}{2},\ \Delta'_{i_1,\,i_1-1} = \frac{\Delta_{i,\,i-1}}{2},\ \Delta'_{i_1,\,i_2} = \frac{\Delta_{i,\,i-1}}{2},\ \Delta'_{i_1,\,i_n} = \frac{\Delta_{i,\,i+1}}{2}$$

or

$$\Delta'_{i_1,\,i_1+1} = \frac{\Delta_{i,\,i+1}}{2}\ \Delta'_{i_1,\,i_1-1} = \frac{\Delta_{i,\,i-1}}{2}\ \Delta'_{i_1,\,i_2} = \frac{\Delta_{i,\,i-1} + \Delta_{i,\,j}}{4}\ \Delta'_{i_1,\,i_n} = \frac{\Delta_{i,\,i+1} + \Delta_{i,\,k}}{4}$$

Figure 5.11 Non-uniform Doo-Sabin subdivision for meshes with arbitrary connectivity

5.1.2 Catmull-Clark Scheme

The scheme constructed by Ed Catmull and Jim Clark is a generalization of the bi-cubic uniform B-Spline surfaces; the details on this subdivision can be found in the original paper [9]. This subdivision scheme can also be generalized to construct non-uniform bi-cubic B-Spline surfaces, as shown in [75] by Sederberg.

The Catmull-Clark scheme generates limit surfaces that are G^2: locally the surfaces are uniform bi-cubic B-Splines at every point except at a finite number of extraordinary points. The extraordinary points correspond to the vertices with a valence larger than four after one subdivision step. The scheme can be classified as primal and non-interpolating.

The scheme is described by two components:

- *Topological* changes: this scheme inserts new vertices at each edge and at each face of the input mesh. These new vertices and the vertices of the original input mesh are then connected together, splitting the original faces into subfaces, as shown in Figure 5.12.

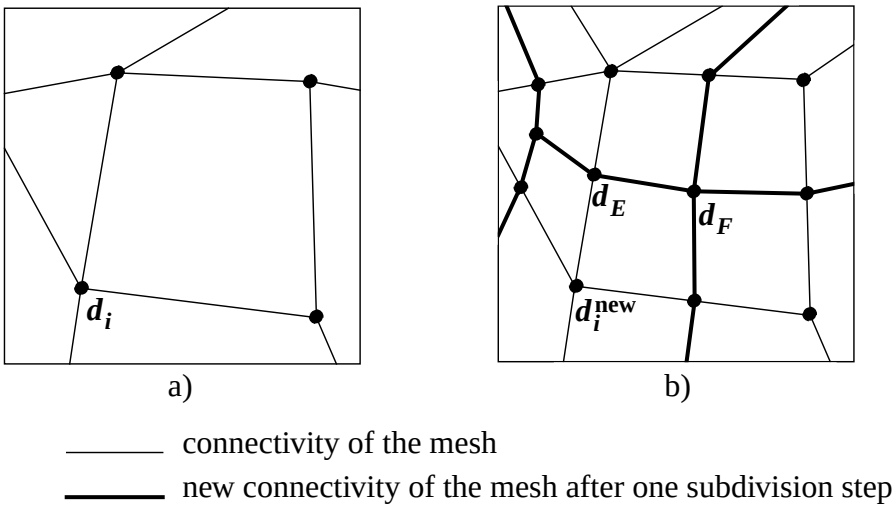


Figure 5.12 Topological component of the Catmull-Clark refinement scheme
a) surface before the refinement step
b) surface after the refinement step

  Since the vertices present in the mesh before the refinement step are not removed during the refinement this scheme is primal.

  It is interesting to see that after one iteration all the faces of the mesh are quadrilaterals.

- *Geometric* changes: In order to describe the geometric position of the vertices after a subdivision step three masks are used: the first mask describes the position of the new face vertices $\boldsymbol{d_F}$, the second describes the position of the new edge vertices $\boldsymbol{d_E}$ and the third describes the new position $\boldsymbol{d_i^{new}}$ of the old vertices $\boldsymbol{d_i}$.

The three masks are presented in Figure 5.13.

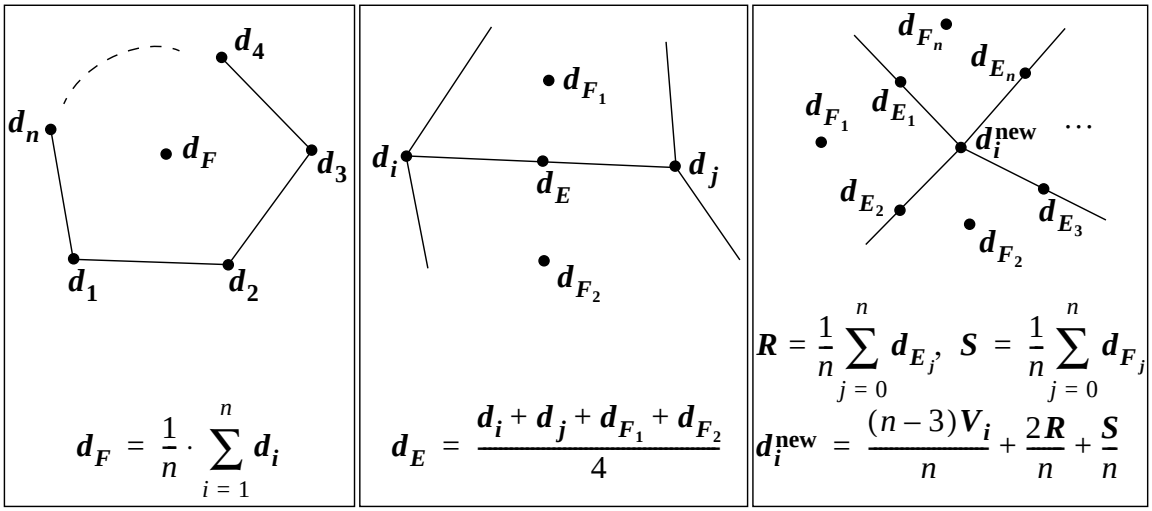


Figure 5.13 Geometric component of the Catmull-Clark refinement scheme
a) The face mask is used to compute the position of the new face vertices
b) The edge mask is used to compute the position of the new edge vertices
c) The vertex mask is used to compute the new position of the old vertices

Applying the Catmull-Clark subdivision to the surface presented in Figure 5.6 results in the meshes shown in Figure 5.14:

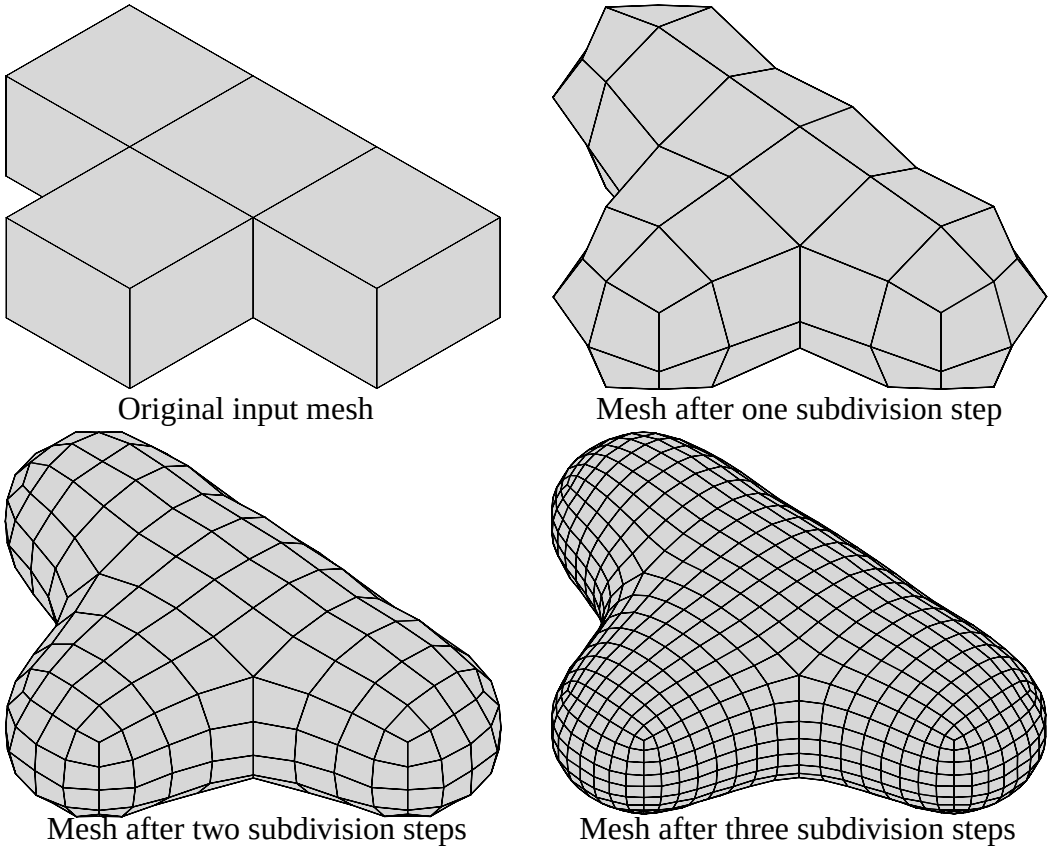


Figure 5.14 Example of the Catmull-Clark subdivision

5.1.3 Loop Scheme

The Loop scheme is based on the subdivision of quartic uniform box splines and it is applied on triangular meshes. A detailed description of box splines can be found in the book of de Boor et al. [21], and the Loop scheme is analyzed in detail in the thesis of C. Loop [55].

The Loop subdivision generates in the limit a G^1 surface: a tangent plane continuous surface everywhere except at a finite number of extraordinary points. The extraordinary points in a limit surface generated by the Loop scheme correspond to the vertices with a valence different from six in the original input mesh. This subdivision can be classified as a non-interpolating primal scheme.

There are two components that have to be described to fully characterize this scheme:

- *Topological* changes: in a refinement step of this scheme subdivides the triangles present in the mesh using the quaternary subdivision rule, as shown in Figure 5.15.

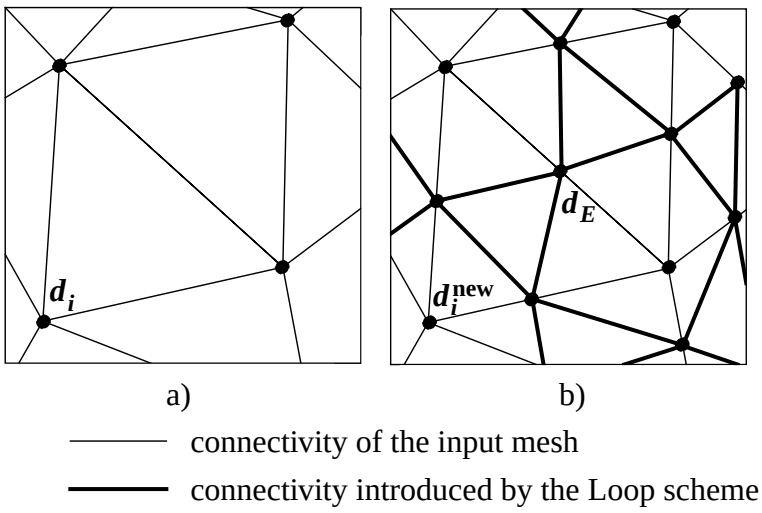


Figure 5.15 Topologic component of the Loop scheme
a) Mesh before the refinement
b) Mesh after the refinement

- *Geometric* changes: the position of the vertices after a subdivision step can be described using two masks: the first mask, shown on the left side of Figure 5.16, describes the new position $\boldsymbol{d_i^{new}}$ of the old vertices $\boldsymbol{d_i}$, and the second mask, shown on

the right side of Figure 5.16, describes the position of the new vertices introduced by the quaternary subdivision rule at the midpoint of every edge $\boldsymbol{d_E}$.

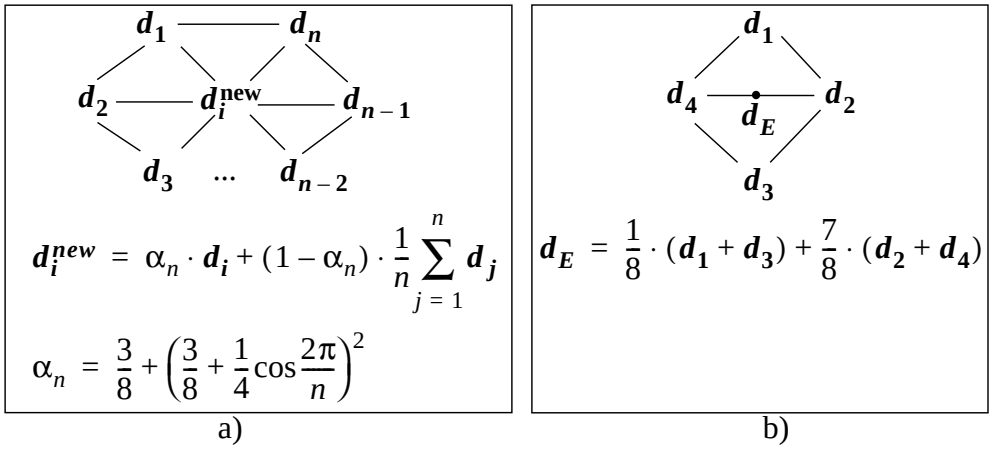


Figure 5.16 Geometric component of the Loop scheme
a) The vertex mask
b) The edge mask

In Figure 5.17 the Loop scheme is applied to a simple mesh. The mesh used in this example is the triangulated surfaces used both in Figure 5.6 and in Figure 5.14.

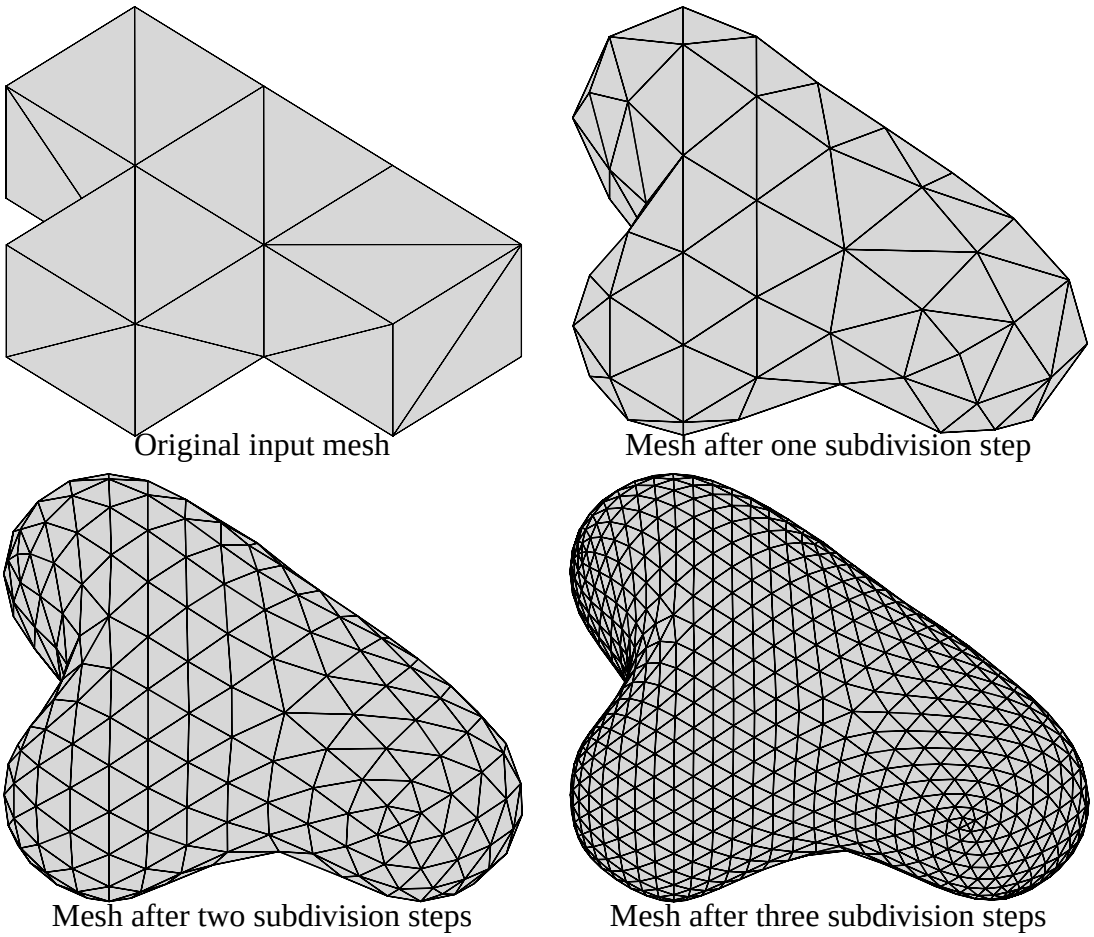


Figure 5.17 Example of the Loop subdivision

5.1.4 Butterfly Scheme

The butterfly scheme has been presented by Dyn, Gregory and Levin in [29] and then refined in [28]; it is an interpolating primal scheme.

This scheme works over regular quaternary subdivision triangular meshes; the resulting limit surface is tangent plane continuous (C^1 continuous).

This scheme is completely characterized by two components:

- *Topological* changes: the butterfly subdivision uses the same rule to refine the microtopology of the input mesh as the Loop scheme. A refinement is computed with the quaternary subdivision rule, as shown in Figure 5.18.

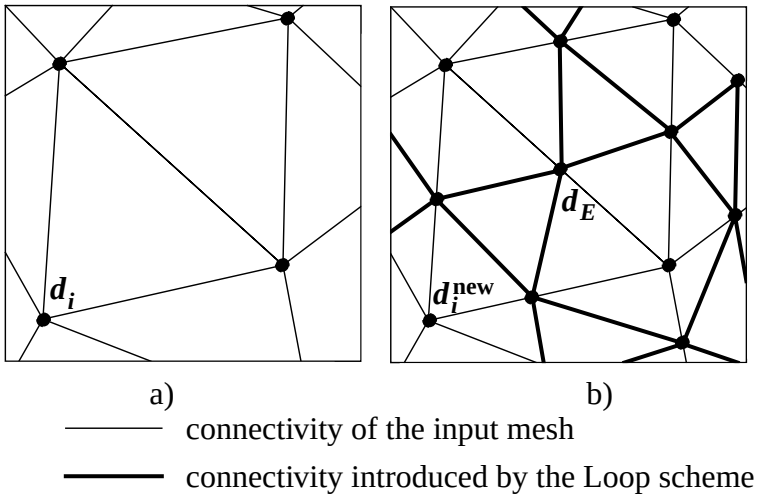
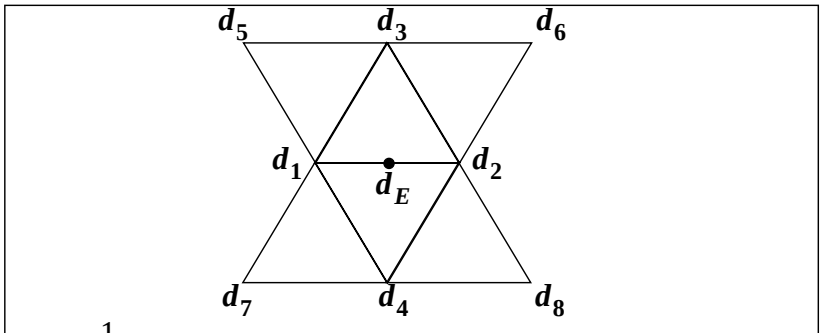


Figure 5.18 Topological component of the Butterfly scheme
a) Mesh before the refinement
b) Mesh after the refinement

- *Geometric* changes: The position of the vertices after a subdivision step can be computed using two masks: the first mask maps the old vertices $\boldsymbol{d_i}$ to the new position $\boldsymbol{d_i^{new}}$, and the second mask describes the geometric position new vertices $\boldsymbol{d_E}$ introduced by the quaternary subdivision scheme. The two masks are shown in Figure 5.19.



$$\boldsymbol{d_i^{new}} = \boldsymbol{d_i}$$

$$\boldsymbol{d_E} = \frac{1}{2} \cdot (\boldsymbol{d_1} + \boldsymbol{d_2}) + 2\omega \cdot (\boldsymbol{d_3} + \boldsymbol{d_4}) - \omega \cdot (\boldsymbol{d_5} + \boldsymbol{d_6} + \boldsymbol{d_7} + \boldsymbol{d_8})$$

Figure 5.19 Geometric component of the Butterfly scheme

It is important to understand that this mask converges to a C^1 surface only if the input mesh is a quaternary subdivision triangular mesh. Extensions to the Butterfly scheme, such as the scheme presented in [89], allow to use the interpolatory scheme on irregular meshes.

Since this scheme works only on regular meshes it is able to generate C^1 limit surfaces that are smooth everywhere and do not contain any extraordinary points.

Figure 5.20 illustrates some of the meshes generated with this scheme. The input surface used in this example is the same surface used in Figure 5.17. Note that the input mesh does not have quaternary subdivision connectivity.

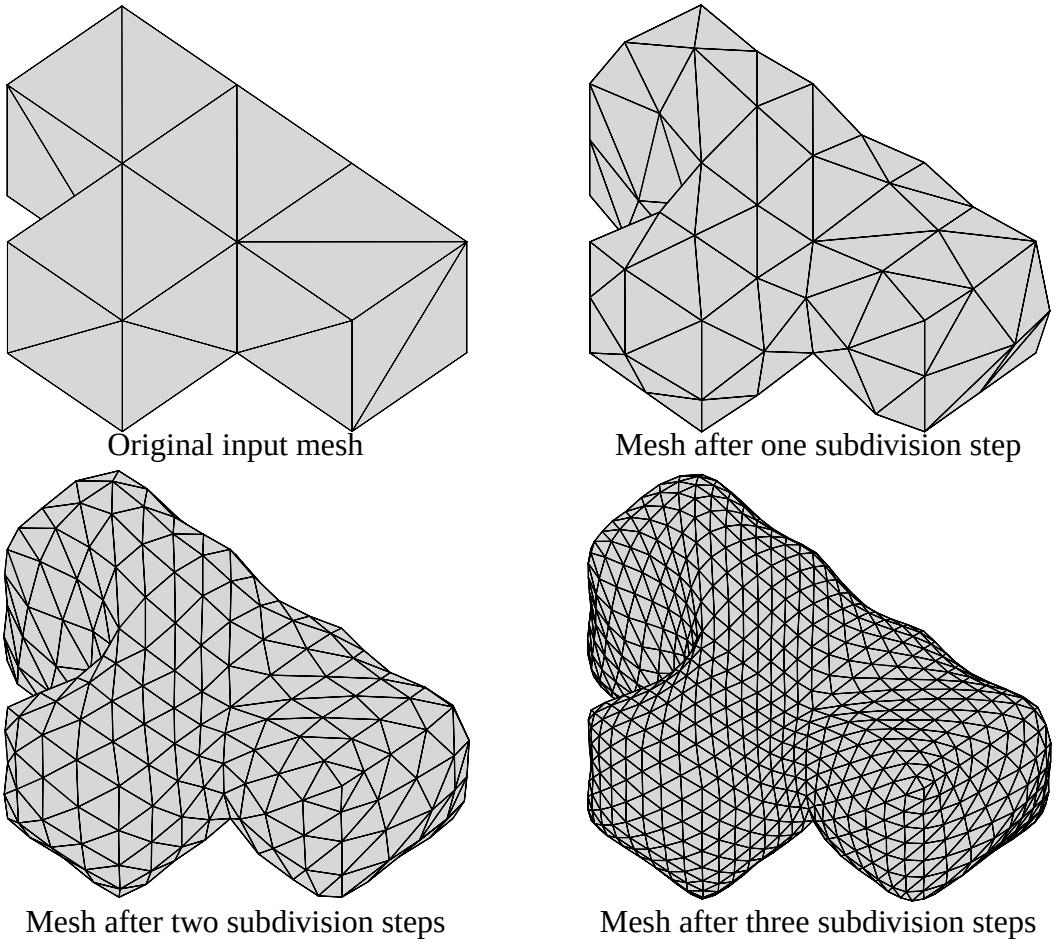


Figure 5.20 Example of the Butterfly subdivision

In Figure 5.20 artifacts can be observed that strongly degrade the quality of the surface. More advanced interpolating subdivisions have removed these artifacts.

5.2 Subdivision Wavelet Transform

The Subdivision Wavelet Transform (SWT) is a technique that makes use of the theory of multiresolution analysis and of the subdivision rules to construct a multiresolution surface representation for surfaces of arbitrary topological type that have subdivision connectivity.

This section introduces the theory behind the SWT. A multiresolution analysis using subdivision surfaces can be constructed using the following steps:

- Construction of refinable scalar basis function
- Construction of nested linear spaces
- Construction of an inner product
- Construction of the wavelet function
- Construction of a filter bank algorithm

In the next subsections each of these steps will be described. More details on this representation can be found in [57] and [56]. Eck et al. presented in [30] a re-meshing strategy that allows one to represent any mesh with the subdivision wavelet transform scheme, thus overcoming the limitation imposed by the subdivision connectivity.

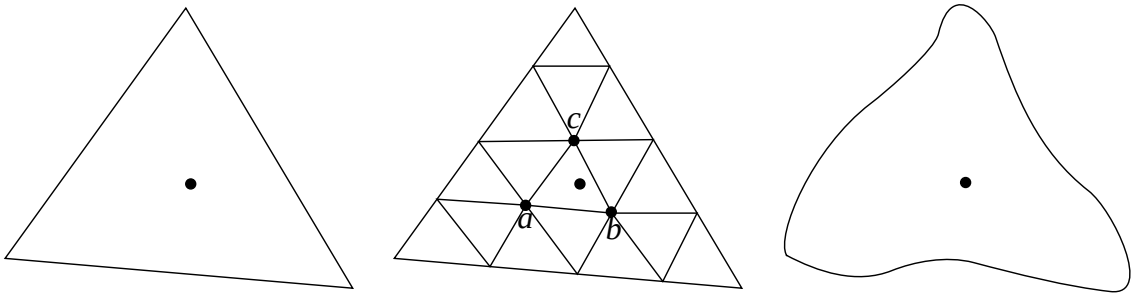
The multiresolution analysis depends on the subdivision scheme used and on the connectivity of the input mesh. In the theory described below we will restrict our attention to triangular meshes and use the *quaternary subdivision* scheme for refinement. The results obtained with these assumptions can be generalized to any subdivision scheme.

5.2.1 Definitions

In order to construct the refinable basis functions that are needed for the multiresolution analysis based on subdivision surfaces, some definitions are helpful:

- $\boldsymbol{M^0}$ stores the topological component of the mesh at the coarsest level 0.
  $\boldsymbol{C^0}$ stores the geometric position of the vertices defined in $\boldsymbol{M^0}$.
- $\boldsymbol{M^s}$ and $\boldsymbol{C^s}$ store the mesh and the vertices generated from the coarse mesh $\boldsymbol{M^0}$ using the subdivision rule s times.
- Next, a parametrization $S(\boldsymbol{x})$ that maps any point $\boldsymbol{x} \in \boldsymbol{M^0}$ to a point on the limit surface is required to construct the basis functions. The parametrization is constructed in three steps:
  1. $S^0(\boldsymbol{x}) = \boldsymbol{x}, \forall \boldsymbol{x} \in \boldsymbol{M^0}$
  2. $S^s(\boldsymbol{x}) = \alpha \cdot \boldsymbol{c_a^s} + \beta \cdot \boldsymbol{c_b^s} + \gamma \cdot \boldsymbol{c_c^s}$, where $\boldsymbol{c_a^s}, \boldsymbol{c_b^s}, \boldsymbol{c_c^s} \in \boldsymbol{C^s}$, and (α, β, γ) defines the barycentric coordinates of the point $\boldsymbol{x}$ in the triangle $\{a, b, c\} \in \boldsymbol{M^s}$.
  3. $S(\boldsymbol{x}) = \lim_{s \to \infty} S^s(\boldsymbol{x})$

These three steps are illustrated in Figure 5.21:



$S^0(\boldsymbol{x}) = \boldsymbol{x}$ a)

$S^s(\boldsymbol{x}) = \alpha \cdot \boldsymbol{c}_a^s + \beta \cdot \boldsymbol{c}_b^s + \gamma \cdot \boldsymbol{c}_c^s$ b)

$S(\boldsymbol{x}) = \lim_{s \to \infty} S^s(\boldsymbol{x})$ c)

Figure 5.21 Parametrization of x
a) Position on the input mesh $\boldsymbol{M^0}$
b) Position on the mesh $\boldsymbol{M^s}$ obtained after s refinement steps
c) Position on the limit surface generated by the subdivision scheme

5.2.2 Construction of the Scalar Basis Function

With the definitions introduced in Section 5.2.1 it is possible to define a refinable basis function, which maps any point $\boldsymbol{x} \in \boldsymbol{M^0}$ to the limit surface.

Lemma 1: *For all $j \geq 0$ and $s \geq j$ there exist a vector of functions $\phi^{s \leftarrow j}(\boldsymbol{x})$ such that*

$$S^s(\boldsymbol{x}) = \phi^{s \leftarrow j}(\boldsymbol{x}) \cdot \boldsymbol{C^j} \quad \text{(EQ 5.10)}$$

Proof 1: Equation 5.10 can be re-written as

$$S^s(\boldsymbol{x}) = \boldsymbol{b^s}(\boldsymbol{x}) \cdot \boldsymbol{C^s} \quad \text{(EQ 5.11)}$$

where the vector $\boldsymbol{b^s}(\boldsymbol{x})$ is defined as

$$b^s(\boldsymbol{x}) = (0 \ldots 0\alpha 0 \ldots 0\beta 0 \ldots 0\gamma 0 \ldots 0) \quad \text{(EQ 5.12)}$$

and α, β, and γ are the barycentric coordinates of the point $\boldsymbol{x}$ as described in Section 5.2.1.

The vertex vector $\boldsymbol{C^s}$ can be written as

$$\boldsymbol{C^s} = P^{s-1} \cdot P^{s-2} \cdot \ldots \cdot P^0 \cdot \boldsymbol{C^0} \quad \text{(EQ 5.13)}$$

where $\boldsymbol{C^0}$ is the geometry of the input coarse mesh at level 0, and P^i is a matrix that maps $\boldsymbol{C^i}$ to $\boldsymbol{C^{i+1}}$. The matrices P^i model the geometric component of the i-th step of the subdivision.

The definitions introduced by Eq. 5.12 and Eq. 5.13 allow us to define the function $\phi^{s \leftarrow j}(x)$:

$$\phi^{s \leftarrow j}(x) = b^s(x) \cdot P^{s-1} \cdot \ldots \cdot P^j \qquad \text{(EQ 5.14)}$$

This lemma can be used to prove the theorem:

Theorem 3: *For any local and uniform convergent continuous subdivision and for any* $j \geq 0$ *a vector of scaling functions* $\phi^j(x)$, $x \in M^0$ *exists such that:*

$$S(x) = \phi^j(x) \cdot C^j \qquad \text{(EQ 5.15)}$$

Proof 2: From the definition of $S(x)$ given in Section 5.2.1 and Lemma 1 it follows

$$\begin{aligned} S(x) &= \lim_{s \to \infty} (\phi^{s \leftarrow j}(x) \cdot C^j) \\ &= (\lim_{s \to \infty} \phi^{s \leftarrow j}(x)) \cdot C^j \end{aligned} \qquad \text{(EQ 5.16)}$$

From which the scalar basis function $\phi^j(x)$ is extracted

$$\phi^j(x) = \lim_{s \to \infty} \phi^{s \leftarrow j}(x) \qquad \text{(EQ 5.17)}$$

A direct consequence of Lemma 1 and Theorem 3 is stated in

Theorem 4: *The scalar functions in* $\phi^j(x)$ *are refinable*

Proof 3: The proof follows directly from Eq. 5.17 and Eq. 5.14:

$$\phi^j(x) = \phi^{j+1}(x) \cdot P^j \qquad \text{(EQ 5.18)}$$

5.2.3 Construction of the Nested Linear Spaces

The linear space $V^j(M^0)$ associated with a mesh M^0 is generated by the span of $\phi^j(x)$:

$$V^j(M^0) = \text{span}\,(\phi^j(x)) \qquad \text{(EQ 5.19)}$$

This definition is equivalent to the definition of a linear space V^j in classic wavelet theory, but its interpretation is more complex: Eq. 5.19 can be interpreted as the space of all the limit surfaces that can be generated by the subdivision rule that uses the coarse mesh M^0 as input.
The space $V^j(M^0)$ can be constructed either by computing the limit surfaces generated by the space of the geometric vectors C^j or equivalently by computing the span of $\phi^j(x)$ over the space of the geometric vectors C^j.

From the definitions of the linear spaces given in Eq. 5.19 and Theorem 4 it follows that these linear spaces are nested:

$$V^0(\boldsymbol{M^0}) \subset V^1(\boldsymbol{M^0}) \subset \ldots \subset V^j(\boldsymbol{M^0}) \subset \ldots \quad \text{(EQ 5.20)}$$

5.2.4 Definition of the Inner Product

The *inner product* of functions in $V^j(\boldsymbol{M^0})$ is another essential component of a multiresolution analysis. For the SWT the inner product between two functions $f, g \in V^j(\boldsymbol{M^0})$ is defined as:

$$\langle f, g \rangle = \int_{\boldsymbol{x} \in \boldsymbol{M^0}} f(\boldsymbol{x}) \cdot \overline{g(\boldsymbol{x})} d\boldsymbol{x} \quad \text{(EQ 5.21)}$$

For simplicity $d\boldsymbol{x}$ is set to the area of a triangulation homeomorphic to $\boldsymbol{M^0}$ consisting of equilateral triangles with edges of length one.

Since the scalar basis function $\phi^{\boldsymbol{j}}(\boldsymbol{x})$ is only defined in the limit, the limit surfaces generated by a subdivision scheme do not have a closed form. Therefore it seems impossible to compute the exact solution of Eq. 5.21, and that the solution can only be approximated using numerical integration algorithms. This is not the case: most subdivisions only use local subdivision rules for the refinement, and in these cases it is possible to compute the inner product exactly.

There is no unique strategy to evaluate the inner product Eq. 5.21, since the computations depend on the subdivision rules, but Lounsbery outlines a general strategy:

1. The first step consists in re-writing Eq. 5.21.
   Since the two functions f, g both belong to $V^j(\boldsymbol{M^0})$ they can be expressed as

$$\begin{aligned} f(\boldsymbol{x}) &= \phi^{\boldsymbol{j}}(\boldsymbol{x}) \cdot F \\ g(\boldsymbol{x}) &= \phi^{\boldsymbol{j}}(\boldsymbol{x}) \cdot G \end{aligned} \quad \text{(EQ 5.22)}$$

   where both F and G are vertex vectors. Eq. 5.21 can now be rewritten as

$$\langle f, g \rangle = \int (\phi^{\boldsymbol{j}}(\boldsymbol{x})F)^T \cdot \phi^{\boldsymbol{j}}(\boldsymbol{x})G \, d\boldsymbol{x} = F^T (\int (\phi^{\boldsymbol{j}}(\boldsymbol{x}))^T \cdot \phi^{\boldsymbol{j}}(\boldsymbol{x}) \, d\boldsymbol{x}) G = F^T L^j G \quad \text{(EQ 5.23)}$$

   where $(L^j)_{i,k} = \langle \phi_{\boldsymbol{i}}^j(\boldsymbol{x}), \phi_{\boldsymbol{k}}^{\boldsymbol{j}}(\boldsymbol{x}) \rangle$, and $\phi_{\boldsymbol{i}}^{\boldsymbol{j}}(\boldsymbol{x})$ is the i-th element of the vector $\phi^{\boldsymbol{j}}(\boldsymbol{x})$.

2. Once the matrix L^j has been computed it is possible to compute the matrix L^{j-1} by making use of Eq. 5.18:

$$L^j = \int (\phi^{\boldsymbol{j}}(\boldsymbol{x}))^T \cdot \phi^{\boldsymbol{j}}(\boldsymbol{x}) = \int (P^j)^T (\phi^{\boldsymbol{j+1}}(\boldsymbol{x}))^T \cdot \phi^{\boldsymbol{j+1}}(\boldsymbol{x}) P^j = (P^j)^T L^{j+1} P^j \quad \text{(EQ 5.24)}$$

   Eq. 5.24 implies that once the matrix L^n for the finest level n of the input surface has been computed, then all the other matrices $L^k, k < n$ can be computed with two matrix-

matrix multiplications, which can be computed efficiently, since the matrix P^j is sparse.

3. Finally, the entries of the matrix L^n at the finest level n have to be computed. There is no fixed rule to compute these entries, but using the locality of the subdivisions it is possible to construct a system of equations whose solution corresponds to the entries of L^j: for example $\langle \phi_i^j(x), \phi_k^j(x) \rangle$ is zero if the support of the i-th scalar basis function does not intersect the support of the k-th scalar basis function.

5.2.5 Construction of the Wavelet Basis Function

In order to define the wavelet basis functions some notation must be introduced: the scaling functions for any primal subdivision rule can be written as

$$\phi^j(x) = (O^j(x), N^j(x)) \qquad \text{(EQ 5.25)}$$

where $O^j(x)$ consists of all scaling functions associated with the old vertices of M^j, and $N^j(x)$ consists of all scaling functions associated with the new vertices introduced by the refinement from M^j to M^{j+1}. In a similar way it is possible to write the matrix P^j as

$$P^j = (\mathrm{O}^j, \mathrm{N}^j) \qquad \text{(EQ 5.26)}$$

where O^j is the submatrix that maps the old vertices of M^j to their new position, and N^j is the submatrix that define the position of the new vertices introduced by the subdivision.

The wavelet basis function is built indirectly by first constructing a basis for $V^{j+1}(M^0)$:

$$\{\phi^j(x), N^{j+1}(x)\} \qquad \text{(EQ 5.27)}$$

where $\phi^j(x)$ is the basis for $V^j(M^0)$, and $N^{j+1}(x)$ is defined in Eq. 5.25. One of the obvious requirements for this basis to exist is that the matrix $O^j(x)$ has to be invertible.

$N^{j+1}(x)$ is not admissible yet as a wavelet basis, since in general it is not orthogonal to $\phi^j(x)$, but can be written as

$$N^{j+1}(x) = \psi^j(x) + \phi^j(x)\alpha^j \qquad \text{(EQ 5.28)}$$

The unknown α^j in Eq. 5.28 can be obtained by solving

$$\langle \phi^j(x), \phi^j(x) \rangle \alpha^j = \langle \phi^j(x), N^{j+1}(x) \rangle = (P^j)^T \langle \phi^{j+1}(x), N^{j+1}(x) \rangle \qquad \text{(EQ 5.29)}$$

Eq. 5.29 is obtained from Eq. 5.28 using the definition Eq. 5.18, and observing that $\phi^j(x)$ and $\psi^j(x)$ are orthogonal.

The solution of Eq. 5.29 is then used to compute the value of $\psi^j(x)$:

$$\psi^j(x) = N^{j+1}(x) - \phi^j(x)\alpha^j \quad \text{(EQ 5.30)}$$

5.2.6 Filter Bank Algorithm

The final step of this multiresolution analysis is the construction of a filter bank algorithm that decomposes an input mesh into a set of scalar and wavelet values. It has already been discussed in Section 4.1.3 that four matrices are needed to build a filter bank algorithm:

1. Two analysis matrices A^j and B^j used in the decomposition step.
2. Two synthesis matrices P^j and Q^j used in the reconstruction step.

The synthesis and analysis equations are given in Eq. 5.31:

$$\begin{aligned}(\phi^j(x), \psi^j(x)) &= \phi^{j+1}(x) \cdot (P^j, Q^j) \\ (\phi^j(x), \psi^j(x)) \cdot \begin{pmatrix} A^j \\ B^j \end{pmatrix} &= \phi^{j+1}(x)\end{aligned} \quad \text{(EQ 5.31)}$$

The synthesis filters are then constructed using Eq. 5.18, Eq. 5.25, Eq. 5.30, and Eq. 5.31:

$$(P^j, Q^j) = \begin{pmatrix} O^j & -O^j\alpha^j \\ N^j & 1 - N^j\alpha^j \end{pmatrix} \quad \text{(EQ 5.32)}$$

where O^j and N^j have been defined in Eq. 5.26.

The two analysis matrices are obtained from the inverse of the synthesis filters computed in Eq. 5.32:

$$\begin{pmatrix} A^j \\ B^j \end{pmatrix} = (P^j, Q^j)^{-1} \quad \text{(EQ 5.33)}$$

5.2.7 Problems of the Subdivision Wavelet Transform Representation

The theory presented in this section briefly describes the multiresolution analysis for the Subdivision Wavelet Transform; most of the proofs are constructive and it is possible to build the SWT for a specific subdivision scheme.
What was not discussed is the complexity of the operations needed to compute the SWT. Most of the details can be found on the two papers of Lounsbery; we will just mention the two most important problems that increase the complexity of the representation:

1. The evaluation of α^j in Eq. 5.29 requires the inversion of the matrix L^j. Although this matrix is sparse its inverse is not, and therefore the evaluation of α^j has a complexity of $O(n^2)$.

2. The synthesis filters are sparse if the subdivision rule has a compact local support, but their inverse, the analysis filters, do not have a compact support in general. This implies that the decomposition step has a total complexity $O(n^2)$.

These problems have been investigated by Lounsbery, who proposes some possible solutions. The complexity of the SWT can be reduced by using “almost” orthogonal wavelets.

5.2.8 An Example of the SWT Representation

In this section we will show a simple example of the subdivision wavelet transform representation using the most simple subdivision rule, which is defined as:

- The topology is refined using the quaternary subdivision rule
- The position of the old vertices is interpolated, and the position of the new vertices is set at midpoint of the old edges.

Figure 5.22 shows a SWT refinement step: the input mesh $\boldsymbol{M^0}$ is first transformed into the mesh $\boldsymbol{M^1}$ using the subdivision operator ($\phi^{j+1}(\boldsymbol{x}) \cdot P^j$ in Eq. 5.31), and then the detail values are added back to the surface ($\phi^{j+1}(\boldsymbol{x}) \cdot Q^j$ in Eq. 5.31).

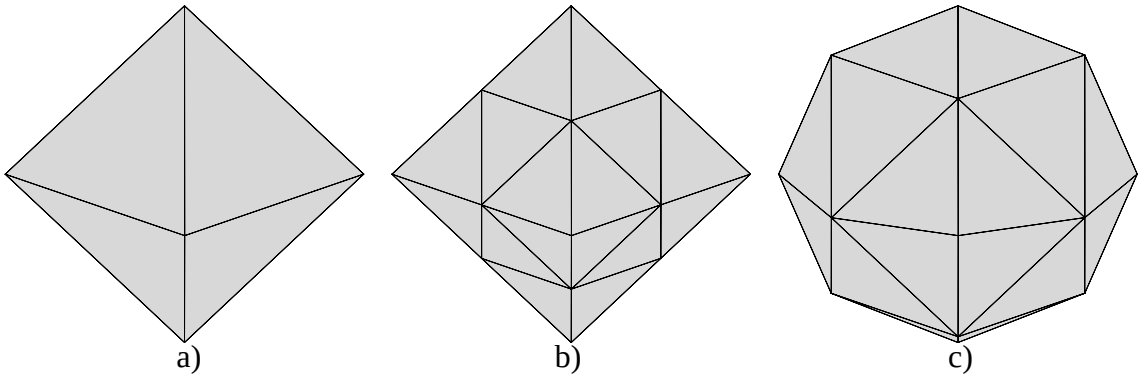


Figure 5.22 The subdivision wavelet transform
a) surface at level 0
b) surface at level 1 after subdivision
c) surface at level 1 reconstructed completely

5.3 Evaluation

In this section the basic subdivision schemes presented in Section 5.1 as well as the subdivision wavelet transform presented in Section 5.2 will be evaluated using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: the representations based on subdivision that are described in this section are capable of generating two-manifold surfaces; the macrotopology of the mesh is not restricted, but the meshes that are generated have subdivision connectivity.

  The *basic subdivision schemes* do not introduce any new feature in the mesh, they only refine the mesh and smooth the vertices, until in the limit a smooth surface is generated.

When constructing the coarse mesh that will be fed to the subdivision algorithm it is therefore crucial to include all the features that need to be present in the surface.
The *subdivision wavelet transform* extends the basic scheme to a multiresolution representation, where the limitation described above does not apply. The SWT algorithm starts from a full resolution mesh that has subdivision connectivity, and using coarsification algorithms it builds the representation. Since the algorithm works with the full resolution surface all the features will be present and can be modeled.

The subdivision rules described in this section have one serious limitation: they are only defined in the interior of the meshes, and they define no rule to refine the boundaries of the meshes. Some papers address this problem and try to construct viable solution to smooth the boundaries of a surface, see for example the work of A. Levin and D. Zorin in [53], [54] and [88].

- *Computation of surface-surface intersections*: As discussed in Section 4.5 there are two different approaches to the computation of surface-surface intersections:

  1. A hierarchical bounding box data structure can be stored to speed up the computation of the intersection. The data structure is separated from the data structure of the representation, and it is only used to resolve intersections.

  2. It can be argued that algebraic methods could be applied on subdivision surfaces easily, since the limit surface of the Doo-Sabin and Catmul-Clark subdivisions generate in the limit bi-quadratic or bi-cubic B-Spline surfaces everywhere except at a finite number of extraordinary points. The problem with this representation is that in general meshes with arbitrary connectivity do not have a global parametrization. Furthermore the implicitization of the spline function in this setting is very complex, if at all possible. It can be concluded that algebraic methods are not a viable solution at the moment for subdivision surfaces.

- *Scalable representation*: The subdivision operators of the schemes defined in Section 5.1 are by definition a refinement operator. Applying a subdivision step to a mesh will increase the number of vertices in the mesh. The *basic subdivision* is therefore not a scalable representation of surfaces.

  The *SWT* representation constructs a multiresolution analysis based on the operators defined in Section 5.1. The resulting framework is therefore scalable. The properties of the SWT are outlined below:

  - The subdivision wavelet transform allows to construct multiresolution representations of meshes that possess subdivision connectivity. The representation is compact: the geometric information can be stored without overhead, and the topologic information need only be stored for the coarse level mesh. The topology of the finer levels of the mesh do not need to be stored explicitly, since it is based on the regular subdivision operator, such as quaternary subdivision.

  - A full resolution mesh can be approximated by starting from the coarse representation of the mesh and by adding as many of the detail values as needed to achieve a certain quality. Furthermore if orthogonal wavelets are constructed, it is possible to build best approximations to the full resolution mesh simply by leaving out the k details with smallest absolute value.

  - It has been discussed in Section 5.2.7 that the construction of the representation might have a high complexity of $O(n^2)$. This happens because the coarsification operator can have a global support on the mesh. It is however possible to construct wavelets that are only "almost" orthogonal much faster.
    The reconstruction algorithm is much faster and has a linear complexity.

- *Modeling of non-manifold singularities, tears and cracks*: both *basic subdivision schemes* and the *SWT* cannot model non-manifolds, since the subdivision operators are only defined over manifold surfaces. In order to model non-manifold surfaces correctly it would be necessary to construct special operators.
  It is very difficult to embed curves in the limit surface, but it would be easy to embed curves in the surface obtained after n subdivision steps, since the resulting mesh would still be piecewise linear. Embedded curves would therefore need to be stored in a separate data structure.

- *Error modeling*: The first few papers on *subdivision* surfaces did not have a strong notion of error, and it was not well understood how to compute the position of a point $\boldsymbol{x}$ on the limit surface. Recent papers on subdivision surfaces built methods to map points from any level to the limit surface using eigenanalysis, and this would help to construct better error models.

  The construction of rigorous errors for the *SWT* is more difficult. As already discussed the error associated with subdivision schemes is well understood, and the wavelet component of the SWT explicitly defines an inner product. One of the components that is missing for an error analysis are well defined basis functions. In Section 5.2.2 and in Section 5.2.5 the scalar and wavelet basis functions have been defined only as a limit toward infinity.

- *Smoothness of the surface*: The different subdivision operators generate different surfaces in the limit, but all of them try to generate smooth surfaces. The better-known operators presented in Section 5.1 either generate G^1 or G^2 surfaces.

  The *SWT* representation is based on the subdivision operators, and it can therefore generate smooth representations. If the full resolution surface is not smooth it is sufficient to compute more subdivision steps to get a smoother surface.

- *Multiresolution editing*: The *basic subdivision schemes* do not define a multiresolution representation of meshes, they just provide operators to refine and smooth meshes. These operators are applied to the input coarse mesh, and in the limit they generate smooth surfaces. It is therefore not possible to compute multiresolution edits on these schemes.
  It is however possible to construct separate data structures that store the edits, which could then be integrated during the subdivision steps. Such an implementation would result in smooth edits, since subdivisions always generate smooth surfaces independently of the input mesh.

  The *SWT* provides a natural framework for edits at different levels of resolution, since the meshes are represented in a multiresolution hierarchy. The edit operations will be smooth, since the underlying basis functions are the subdivision operators.

- *Surface fitting*: In his thesis [42] Hoppe describes a strategy to fit a subdivision surface through a cloud of unorganized points, for which nothing is known, not even the macro-topology of the surface that they define. This is accomplished by first fitting a triangular mesh through the points, then by simplifying it, and finally by building the representation based on a subdivision operator.

  There is no automatic fitting procedure which constructs a SWT representation. It is however possible to use Hoppe's thesis to construct a triangular mesh with subdivision connectivity and then to construct the wavelet representation on top of it.

- *Support of local high variation in the curvature of the surface*: Although the basic subdivision operators described in this section generate smooth surfaces, it is possible to extend them to model non-smooth regions. There are three main contributions that allow us to model non-smooth regions:

  1. In his thesis [42] Hoppe describes an extension of the Loop subdivision that allows to define edges in the mesh that do not have to be smoothed.
  2. T. Sederberg describes a *Non-Uniform Recursive Subdivision Scheme* (NURSS) that extend the classic Doo-Sabin and Catmul-Clark schemes to non uniform B-Splines. This extension has been described in Section 5.1.1; further information can be found in the original paper [75].
  3. T. DeRose presented in [22] a simple strategy that allows to model non-smooth regions with subdivisions.

  The SWT representation can also be extended to support high variation in the curvature, but it would be necessary to construct the appropriate basis functions and matrices for the filter bank algorithm.

- *Changes of the surface over time*: The problem of encoding changes of the mesh over time for the basic subdivision schemes and for the subdivision wavelet transform reduces to the problem of representing the changes in the coarsest input mesh over time. If this can be accomplished, then both representations can automatically handle changes over time.

6 Hierarchical B-Splines

Hierarchical B-Splines are yet another possible surface representation for geologic data. Traditional approaches that use control vertices and locally supported basis functions represent a surface as

$$S(u,v) = \sum_i \sum_j \boldsymbol{d}_{\boldsymbol{i},\boldsymbol{j}} \cdot N_i^k(u) \cdot N_j^l(v) \qquad \text{(EQ 6.1)}$$

where $\{\boldsymbol{d}_{\boldsymbol{i},\boldsymbol{j}}\}$ is a set of control vertices, $N_i^k(u)$ and $N_j^l(v)$ are basis functions with a local support, k and l represent the order of the basis functions, and u and v define the global parametrization of the surface.

The problem with this representation is that either:

1. There are not enough control vertices to model all the desired surface.
2. There are too many control vertices that are not used to model anything.

The hierarchical B-spline approach starts with a sparse set of control vertices and adds more control points locally where they are needed to model small details.

In the next subsections the tools necessary to build a hierarchical B-Spline representation will be developed, namely:

- a refinement operator that increases the number of control points in Eq. 6.1. Increasing the number of control points decreases the support of the basis functions. This allows to introduce small details to the surface.
- an operator that allows to apply a refinement locally on the surface. Next, the concept of *overlay* will be introduced, that will be used to store fine level patches relative to the coarser level patches. This will allow to propagate edit operations from the coarse patches to the fine patches.

All the details regarding hierarchical splines and some of the applications/enhancements can be found in the original papers [36], [35], and [34].

6.1 Refinement Operator

The hierarchical structure presented in this section works with any surface representation that models a surface using locally supported basis functions. To better illustrate the behaviour of the H-Spline representation the cubic B-Spline function will be used as an example. Operators for other basis functions can be computed similarly.

The input of the refinement operator is a surface defined as

$$S(u,v) = \sum_i \sum_j \boldsymbol{d}_{\boldsymbol{i},\boldsymbol{j}} \cdot N_i^k(u) \cdot N_j^l(v) \qquad \text{(EQ 6.2)}$$

The number of control points in the surface can be increased by decreasing the support of the basis function. The surface would then be defined as

$$S(u, v) = \sum_i \sum_j \boldsymbol{d}'_{\boldsymbol{i},\boldsymbol{j}} \cdot N_i^k(2 \cdot u) \cdot N_j^l(2 \cdot v) \qquad \text{(EQ 6.3)}$$

The shape of the new surface described in Eq. 6.3 must remain the same as the shape of the original surface defined in Eq. 6.2. This can be achieved easily, since a B-Spline function $N_i^k(u)$ can be written as a linear combination of B-Spline functions $N_{i-l}^k(2 \cdot u)$:

$$N_i^k(u) = 2^{-(k-1)} \cdot \sum_{j=0}^{k} \binom{k}{j} \cdot N_{i-k+j}^k(2u) \qquad \text{(EQ 6.4)}$$

An example is shown in Figure 6.1:

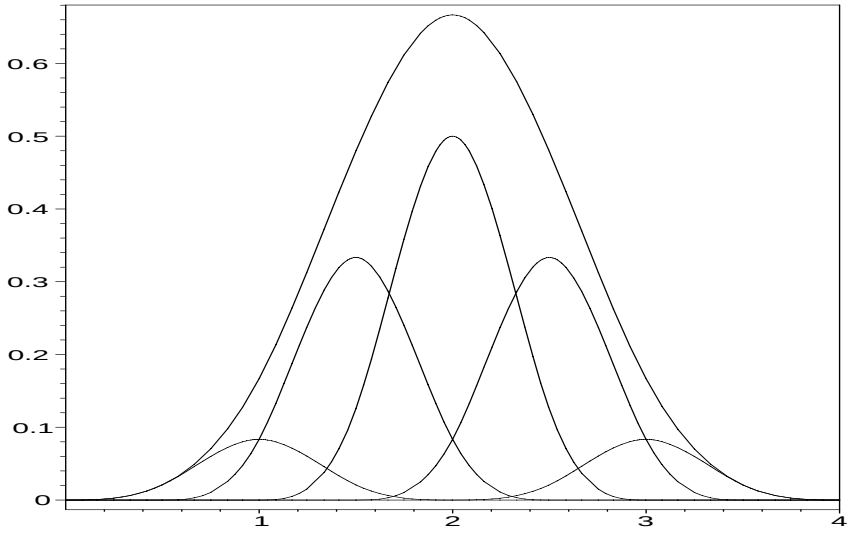


Figure 6.1 Two scale relation for cubic B-spline functions

From Eq. 6.2, Eq. 6.3, and Eq. 6.4 the value of the new control points $\boldsymbol{d}'_{\boldsymbol{r},\boldsymbol{s}}$ can be computed using Eq. 6.5

$$\boldsymbol{d}'_{\boldsymbol{r},\boldsymbol{s}} = \sum_{i=0}^{m} \sum_{j=0}^{n} \alpha_i^k(r)\alpha_j^l(s)\boldsymbol{d}_{\boldsymbol{i},\boldsymbol{j}} \qquad \text{(EQ 6.5)}$$

where the value of $\alpha_i^k(r)$ and $\alpha_j^l(s)$ is derived directly from Eq. 6.4.

In matrix notation, a refinement of the control vertices is specified as

$$D' = \alpha_{\text{left}} \cdot D \cdot \alpha_{\text{right}} \qquad \text{(EQ 6.6)}$$

The equations derived in this section enable us to refine a grid of control points into a finer resolution grid. This, as already mentioned, enables us to create more detailed surfaces.

The operation described in this subsection refines the whole surface. This could introduce a large overhead if the surface has large flat regions, since only few control points are required to model these regions. A straightforward improvement to the representation consists in building operators capable to refine the surface only locally, thus minimizing the number of unused control points.

Forsey was the first to use the refinement formula Eq. 6.4 to construct a multiresolution representation of a surface. This idea was used some years later to construct a wavelet representation that uses endpoint interpolating cubic B-Spline basis functions.

6.2 Offset Referenced Overlays

In the previous section a general subdivision rule for B-Spline surfaces was developed. If this subdivision rule is applied to all the control points of a two-dimensional surface, then the number of control points would increase by four. This is usually not desirable, since only some regions in the surface need the additional degrees of freedom.

We will show an example of how to build a local overlay. The white squares in Figure 6.2 represent the area of influence of the control vertex $\boldsymbol{d}_{i,j}$: if this control points is moved, the surfaces enclosed in the white rectangles will move with $\boldsymbol{d}_{i,j}$.

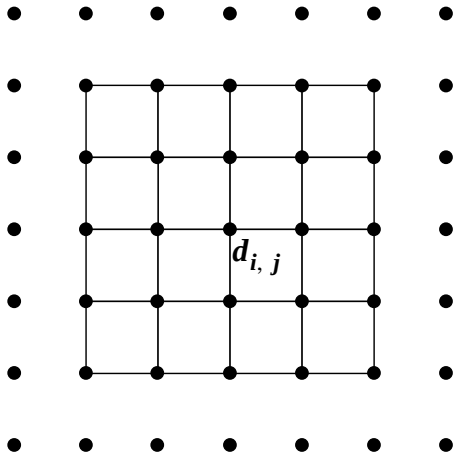


Figure 6.2 The support of a control vertex of a cubic B-Spline surface

If the area of influence of the control vertex is too large, and a users wants to insert finer level details into the surface, then it is necessary to compute a refinement around $\boldsymbol{d}_{i,j}$. The new set of control points generated in this way, called *overlay*, will have a smaller support, thus allowing to edit smaller parts of the surface. A simple overlay is illustrated in Figure 6.3: the support of the center control point $\boldsymbol{d}'_{i,j}$ of the overlay, drawn as a set of

gray squares in the figure, is now just one fourth of the support of the original control point $\boldsymbol{d}_{i,j}$.

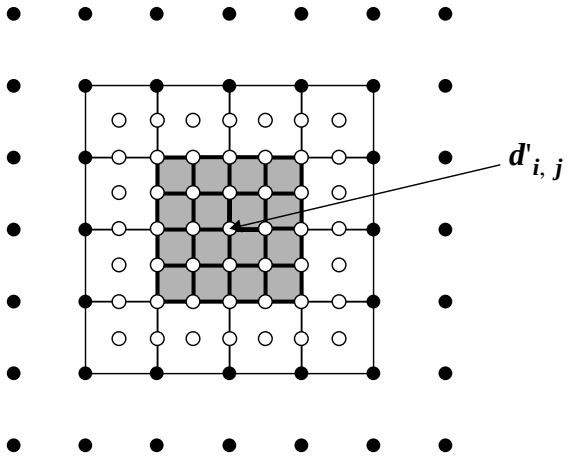


Figure 6.3 An overlay for a cubic B-Spline patch

These overlays can be inserted where needed, so that control points are added only where the complexity of the surface requires them.

There is still one problem that was not addressed: if one of the black control vertices in Figure 6.3 is moved, then this change needs to be propagated to the overlay, since the support of the black vertices and the support of the white vertices of the overlay are not disjoint. If the control vertices $\boldsymbol{d}'_{i,j}$ in the overlay are initialized using Eq. 6.6, then changes in the black control vertices that define the coarse level shape of the surface cannot be propagated to the overlay, and therefore edit operations will not be executed correctly. The author solved this problem by making use of local frames: the control points of the overlay are stored in the form

$$\boldsymbol{d}'_{i,j} = \boldsymbol{r}_{i,j} + \boldsymbol{o}_{i,j} \qquad \text{(EQ 6.7)}$$

where

$$\{\boldsymbol{r}_{i,j}\} = \alpha_{\text{left}} \cdot D \cdot \alpha_{\text{right}} \qquad \text{(EQ 6.8)}$$

stores the initial position of the control vertices dictated by the control vertices in the base mesh, and $\boldsymbol{o}_{i,j}$ stores the edit operations applied to the overlay with respect to a local coordinate system.

6.3 An Example of the H-Spline Representation

In this section we present a small example showing how the H-Spline representation works; we will also show why it is necessary to store the control points of overlays in a local frame.

Figure 6.4 illustrates some edits computed on an surface represented with an H-Spline:

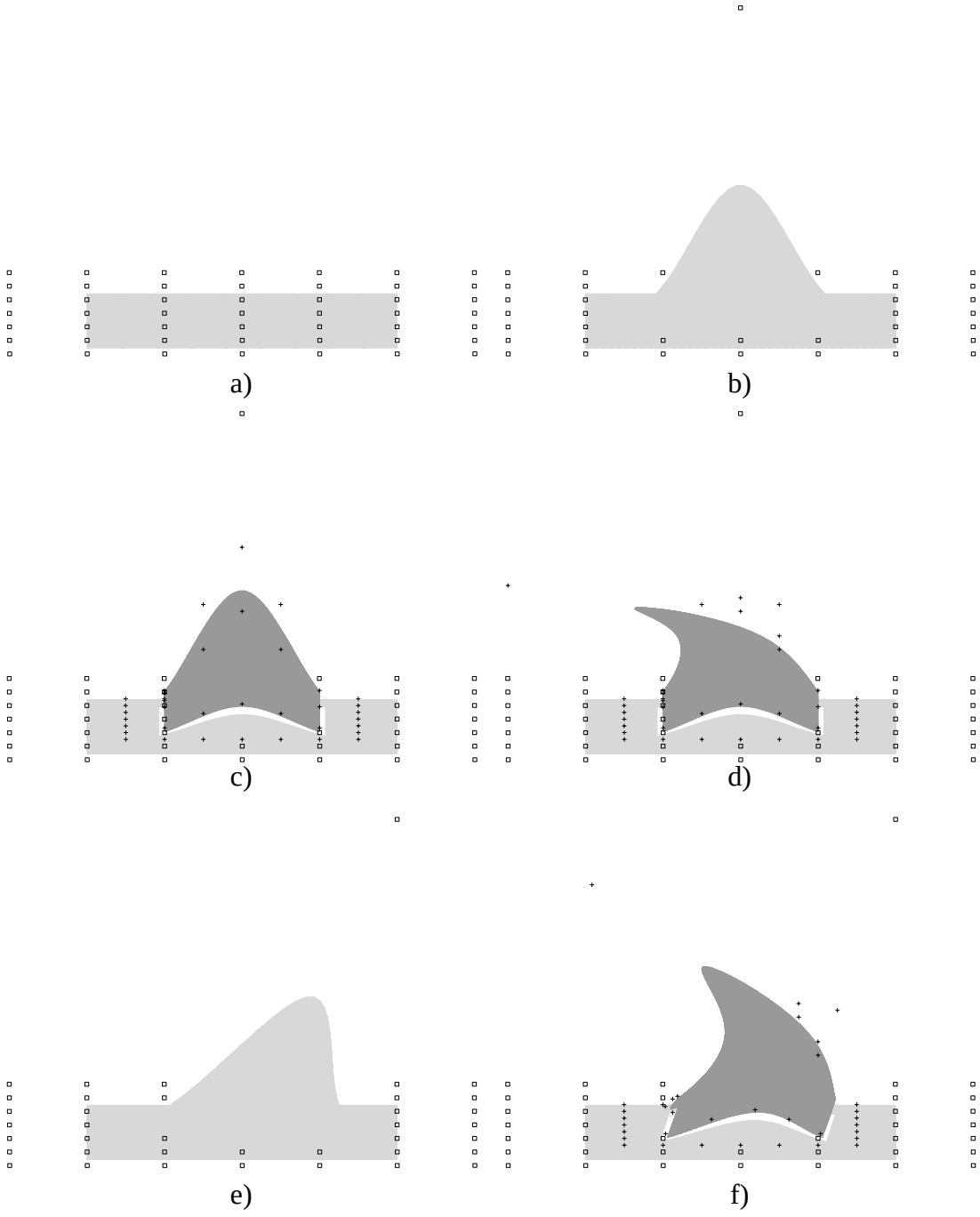


Figure 6.4 Edit of a surface represented as an H-Spline

The next list briefly describes the edit operation presented in Figure 6.4:

- a) represents our initial surface: all the control points lie on a plane, and therefore the surface that they define also lies on a plane.

- In b) the central control point is moved vertically; this results in an edit operation that has support on the whole patch.
- In c) an overlay around the central control point is constructed. The initial position of the control points in the overlay guarantees that the surface in c) interpolates the surface b). The white spaces between the two patches are used to better distinguish between the patches; after the overlay has been inserted the surface remains continuous and smooth.
- In d) the central control point of the overlay is moved horizontally. This does not change the overall shape of the initial patch, but just the dark gray component that describes the support of this control point.
- In e) only the coarse level of the surface is shown. The edit computed in d) is not visible, since it was computed in the overlay. The central control point of the coarse mesh is moved to the right.
- Finally, in the surface shown in f) visualizes the two edit operations computed in d) and e). If the overlay was not stored in a local frame the edits would have not looked natural and the results would have been counterintuitive.

6.4 Evaluation

In this section the hierarchical spline surface representation will be evaluated using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: H-Splines can be used to model complex two-manifold surfaces. Since a global parametrization is required, and since the basic H-Spline algorithm uses uniform splines, it is not possible to model an arbitrary two-manifold surface.
  The H-Splines require a certain regularity, since a global parametrization defined by the two coordinates u and v must exist, but they allow us to model surfaces with irregular boundaries.
- *Computation of surface-surface intersections*: The surface intersection problem could be solved using the two solution already described in Section 4.5 and Section 5.3:
  1. a separate hierarchical bounding box data structure can be used to accelerate the computation of the intersection curves. This data structure stores information on the piecewise linear approximation of the spline that is used for rendering.
  2. an algebraic method can be used to compute more precise solution on the B-Spline surface. This could be accomplished by generalizing existing methods that works on Bezier patches.
- *Scalable representation*: The H-Splines are scalable in the sense that they do not require a uniform refinement in order to model small details in a surface.
  - The representation is compact: an expert user is capable of modeling a surface using the minimum number of control points. Novice users usually generate far too many control points, most of which are not used to model any feature. All these control points could be considered as an overhead.

  - It is easy to extract a good approximation of a surface using a few control points: an approximation can be constructed simply by taking the coarse set of control points and adding any number of overlays.
  - The construction of the representation and the reconstruction of the surface are both very fast algorithms that work in real-time.

- *Modeling of non-manifold singularities, tears and cracks*: In general it is not possible to represent any non-manifold or pseudo-manifold singularities, but some non-manifold surfaces can be easily constructed using B-Splines. For example it is very easy to model a surface that intersect itself, but in general it is not possible to model an horizon or fault that splits into two horizons or faults.

  Embedding of curves is not feasible, since the basic primitive of the H-Splines is the B-Spline, and a curve defined over a B-Spline patch might be represented only with a high order polynomial. Piecewise linear curves can be embedded in the piecewise linear approximation of the surface.

- *Error modeling*: The basis function used as a modeling primitive in the H-Spline representation is the B-Spline. This basis function is very well studied, and it allows to construct advanced error models.

- *Smoothness of the surface*: Depending on the degree of the B-Spline basis function that is chosen to model surfaces, the H-Splines representation is able to model smooth surfaces, for example using cubic B-Splines, or not, for example using the simple order zero B-Splines.

- *Multiresolution editing*: The original paper that introduced the concept of H-Spline described a multiresolution editing tool. This capability is fully integrated in the representation. Edit operations affect the control points of the B-Spline functions and therefore the changes in the surface will be smooth.

- *Surface fitting*: Forsey presented in [34] a technique for an automatic surface fitting of clouds of points. However the surface generated by the algorithm had to be a simple height field.

- *Support of local high variation in the curvature of the surface*: If the underlying B-Spline basis is smooth, then the H-Spline representation will not be able to model non-smooth regions well; if the B-Spline basis is not-smooth, then local high variation in the curvature can be modeled easily.
  In the original concept of the H-Spline it is not possible to model non-smooth regions with higher order splines, since the representation can only use uniform B-Splines. It would be of interest to construct an extension of the H-Spline representation that is based on the non-uniform B-Spline basis functions.

- *Changes of the surface over time*: Changes of the surface over time can be modeled elegantly with H-Splines: it is only necessary to construct three dimensional overlays instead of the standard two dimensional overlays. This strategy would probably introduce some overhead, since overlays might not be needed in the same place at different points in time.

7 Mesh Based Representations

The mesh based surface representations define a surface with a collection of vertices and polygons, usually triangles. This means that these representations only work with piecewise linear surfaces.

In this section some of the better known representation will be described:

- W. Schroeder's vertex removal algorithm [74].
- H. Hoppe's progressive meshes algorithm [43].
- M. Garland's and P. Heckbert's pair contraction algorithm [37].
- Li's compression of 3D models [47].

Closely related to these representations are the papers that apply signal processing techniques to meshes with arbitrary connectivity that will be presented in Section 8.

7.1 Vertex Removal Algorithm

William Schroeder et al. presented in [74] a simple algorithm for mesh decimation based on a vertex removal strategy. The algorithm is general enough to be able to handle non-manifold surfaces, and it allow the user to set constraints to maintain some of the features of the original surface.

The algorithm is subdivided into three steps:

- *characterization of the local topology of a vertex*: each vertex in the mesh is classified according to the microtopology of the neighborhood. The five possible classifications of the vertices are given in Figure 7.1:

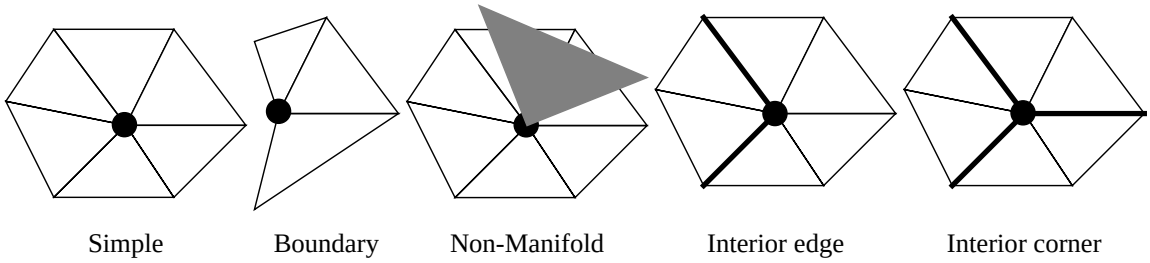


Figure 7.1 Classification of the vertices of a mesh

The first two elements in Figure 7.1 represent the local connectivity common to the vertices of a two-manifold surface: the neighborhood of a vertex is either homeomorphic to a disc, as in the case of *simple*, or to a half disc, as in the case of *boundary.*
In the non-manifold case the surface intersects itself at the vertex, possibly resulting in some edges bounding more than two triangles.
The interior edges and interior vertices are used to constrain the decimation of the surface. Interior edges define the boundary of regions with different properties. For exam-

ple an interior edge could be inserted if the dihedral angle between the two triangles that share the edge is larger than a user-specified threshold.

- *error norm evaluation*: after the first step all the vertices have been classified. In this second step the error introduced by the removal of a vertex must be computed. Schroeder did not use just one error norm, but he constructed different error norms for the different classes of vertices. The following list describes the error norms used in the original algorithm:
  - *Simple vertex*: the error introduced by the removal of a simple vertex $\boldsymbol{v}$ is computed as the distance between $\boldsymbol{v}$ and the average plane p defined by the vertices on the star of $\boldsymbol{v}$, as shown in Figure 7.2.

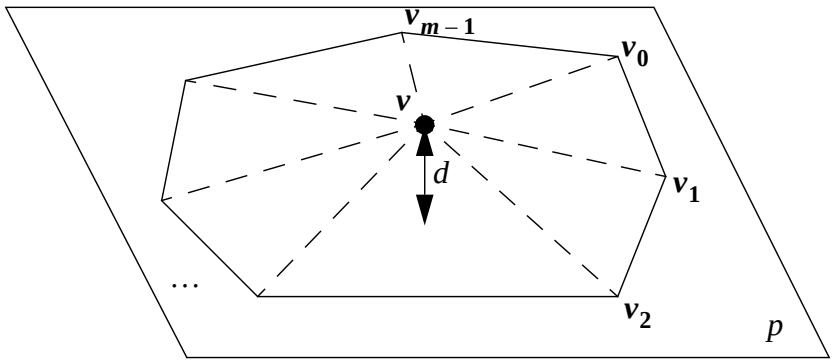


Figure 7.2 Removal of a simple vertex

The distance d can be computed efficiently as

$$d = |\boldsymbol{n} \cdot (\boldsymbol{v} - \boldsymbol{x})| \qquad \text{(EQ 7.1)}$$

where

$$\boldsymbol{n} = \frac{\boldsymbol{N}}{|\boldsymbol{N}|} \qquad \text{(EQ 7.2)}$$

$$\boldsymbol{x} = \frac{\sum_{i=0}^{m-1} \boldsymbol{x_i} \cdot A_i}{\sum_{i=0}^{m-1} A_i} \qquad \text{(EQ 7.3)}$$

and

$$\boldsymbol{N} = \frac{\sum_{i=0}^{m-1} \boldsymbol{n_i} \cdot A_i}{\sum_{i=0}^{m-1} A_i} \qquad \text{(EQ 7.4)}$$

$$\boldsymbol{n_i} = (\boldsymbol{v} - \boldsymbol{v}_{(\boldsymbol{i}+1) \bmod \boldsymbol{m}}) \times (\boldsymbol{v} - \boldsymbol{v_i}) \qquad \text{(EQ 7.5)}$$

$$\boldsymbol{x_i} = \frac{\boldsymbol{v} + \boldsymbol{v_i} + \boldsymbol{v}_{(\boldsymbol{i}+1) \bmod \boldsymbol{m}}}{3} \qquad \text{(EQ 7.6)}$$

$$A_i : \text{area of the triangle defined by } (\boldsymbol{v}, \boldsymbol{v_i}, \boldsymbol{v}_{(\boldsymbol{i}+1) \bmod \boldsymbol{m}}) \qquad \text{(EQ 7.7)}$$

- *Boundary vertices* and *interior edge vertices*: The most important visual artifact introduced by the removal of both boundary and interior edge vertices is not the "flattening" of the geometry, but the change of the shape of the boundary and of the interior edges.
  It makes sense therefore to use a different error norm than the norm defined in Eq. 7.1, such as the error model illustrated in Figure 7.3:

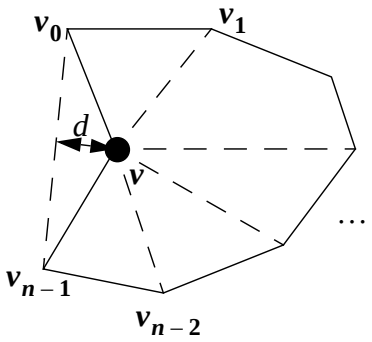


Figure 7.3 Removal of a boundary edge or of an interior edge

  The error norm is therefore proportional to the minimum distance between the vertex $\boldsymbol{v}$ that has been removed and the new edge $(\boldsymbol{v_0}, \boldsymbol{v_{n-1}})$.

- *Non-manifold* and *interior corner vertices*: these two types of vertices are not removed in the decimation process, since their removal could change the topology of the mesh or remove essential features.

The error introduced by the removal of each vertex is stored in a sorted list. The algorithm will then remove the vertex that introduces the smallest error.

- *vertex removal*: in the previous step the vertices have been ranked according to the error that their removal would introduce in the surface. In the next step the vertex that introduces the smallest error is removed from the mesh. After this removal operation the mesh contains a hole that was not present before. It is therefore necessary to re-triangulate the hole.

The algorithm used in the paper is fairly simple, and it is based on a divide-and-conquer strategy. The idea is to connect two vertices on the boundary of the hole with an edge, thus splitting the problem into two sub-problems. Since the algorithm should be capable to handle any non-manifold mesh it is very easy to construct an example where such an algorithm would fail. In this case the vertex is not removed from the mesh.

An example outlining the divide-and-conquer strategy is presented in Figure 7.4:

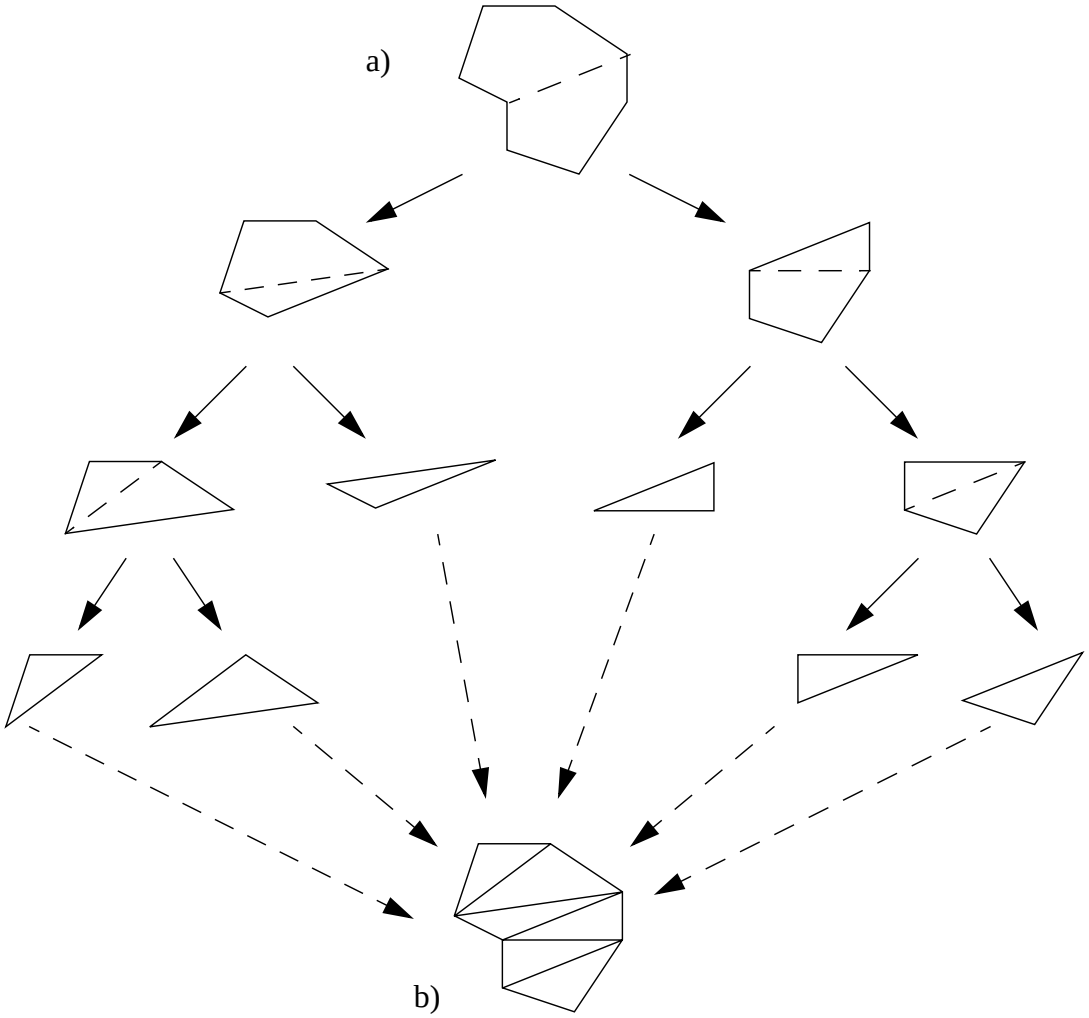


Figure 7.4 Re-triangulation via divide-and-conquer strategy
a) Boundary of the hole that must be triangulated
b) Triangulation of the hole

The error norm evaluation and the vertex removal operations can be applied to simplify the mesh by removing one vertex at a time:

$$\boldsymbol{M} = \boldsymbol{M^n} \rightarrow \boldsymbol{M^{n-1}} \rightarrow \ldots \rightarrow \boldsymbol{M^i} \qquad \text{(EQ 7.8)}$$

After the removal of a vertex v from the mesh the error associated with the vertices on its star must be recomputed, since the removal operation might have changed the error associated with these vertices. The new error values will replace the old values stored in the error list.

7.2 Progressive Mesh

The *progressive mesh* (PM) algorithm by Hugues Hoppe is probably one of the better known mesh based schemes to organize, store and transmit meshes of triangles. A detailed description of this scheme and its various extensions, such as view dependent PM and progressive simplicial complexes, can be found in [43], [44], [67], and [45]. Progressive meshes have been extended to three dimensional tetrahedral meshes in [77].

The progressive mesh scheme builds a continuous level-of-detail approximation of an input mesh $\hat{\boldsymbol{M}}$; the two components that define the representation are:

1. A coarse mesh $\boldsymbol{M^0}$. This mesh is much less detailed than the original mesh and it contains fewer triangles. Usually the mesh $\boldsymbol{M^0}$ corresponds to the coarsest possible mesh that has the same topological type as the input mesh $\hat{\boldsymbol{M}}$.
2. A set of vertex split records vsplit_i, $i = 1, \ldots, n$. Each record vsplit_i contains enough information to split a vertex in two vertices, thus inserting one or two new triangles in the mesh. If a single vertex split operation is applied, it will transform the mesh $\boldsymbol{M^{i-1}}$ to $\boldsymbol{M^i}$. Applying all the n vertex split operations to the mesh $\boldsymbol{M^0}$ will result in the original mesh $\boldsymbol{M^n} = \hat{\boldsymbol{M}}$.

In typical applications the only information available is the input mesh $\hat{\boldsymbol{M}}$: no information is available on the mesh $\boldsymbol{M^0}$ or about the $vsplit_i$ records. It is therefore necessary to develop an algorithm to compute this information.

The necessary information is generated starting from the input mesh $\hat{\boldsymbol{M}}$ applying n edge collapse operations ecol_i, $i = 1, \ldots, n$, which are the inverse of the vsplit_i operations.

These two operations are shown in Figure 7.5.

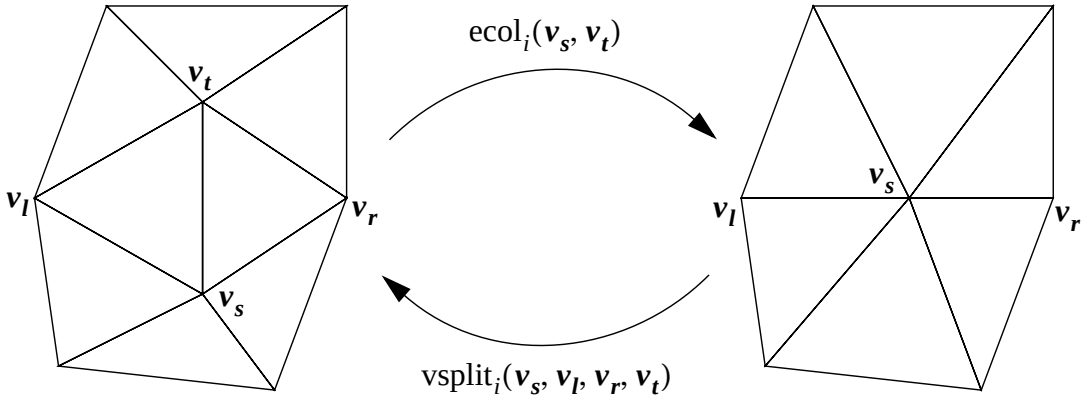


Figure 7.5 Edge collapse and vertex split operations

The information for the progressive mesh scheme is then constructed as follow:

- Starting from the input mesh $\hat{\boldsymbol{M}} = \boldsymbol{M^n}$ perform n edge collapse operations. The resulting mesh is our coarse mesh $\boldsymbol{M^0}$:

$$\hat{\boldsymbol{M}} = \boldsymbol{M^n} \rightarrow \boldsymbol{M^{n-1}} \rightarrow \ldots \rightarrow \boldsymbol{M^0} \quad \text{(EQ 7.9)}$$

- Store the inverse of the $ecol_i$ operations as $vsplit_i$ operations:

$$(ecol_i)^{-1} = vsplit_{n-1-i} \quad \text{(EQ 7.10)}$$

- The last important component required to construct a progressive mesh representation of meshes is an error norm. Eq. 7.9 and Eq. 7.10 specify how to build the representation, and the error norm drives the construction by specifying which vertices should be removed first.
  The error norm will be used to decide how important edges are. This will allow us to compute an edge collapse $ecol_i$ that removes the i-th least important vertex from the mesh. Since least important vertices are removed first, it is possible to construct an error norm that guarantees that the error associated with a mesh $\boldsymbol{M^i}$ is smaller or equal than the error associated with a mesh $\boldsymbol{M^j}$ if $i \geq j$. This is formalized in Eq. 7.11:

$$E(\boldsymbol{M^i}) \leq E(\boldsymbol{M^j}),\ i \geq j \quad \text{(EQ 7.11)}$$

  However the greedy nature of the algorithm does not construct best approximations: in order to guarantee that an approximation satisfies an error too many triangles will be used.

  Furthermore to any approximation $\boldsymbol{M^i}$ there is an associated error $E(\boldsymbol{M^i})$ that specifies how different the approximation is from a reference mesh, usually the full resolution surface.

  The error function that Hoppe has chosen is defined by an energy function:

$$E(\boldsymbol{M^i}) = E_{\text{dist}}(\boldsymbol{M^i}) + E_{\text{spring}}(\boldsymbol{M^i}) \quad \text{(EQ 7.12)}$$

$$E_{\text{dist}}(\boldsymbol{M^i}) = \sum_{l} d^2(\boldsymbol{v_l}, \boldsymbol{M^i}) \quad \text{(EQ 7.13)}$$

$$E_{\text{spring}}(\boldsymbol{M^i}) = \sum_{\{j,k\} \text{ is and edge}} \kappa \left\| \boldsymbol{v_j} - \boldsymbol{v_k} \right\| \quad \text{(EQ 7.14)}$$

  where $\{\boldsymbol{v_l}\}$ are a collection of vertices whose distance $d^2(\boldsymbol{v_l}, \boldsymbol{M^i})$ to the mesh $\boldsymbol{M^i}$ is measured, and κ is an elasticity constant.

  The distance energy described in Eq. 7.13 measures the total squared distance of a collection of points sampled on the reference mesh to the mesh $\boldsymbol{M^i}$. The spring energy function described in Eq. 7.14 corresponds to placing a spring on each edge with tension κ.

The vertices used in the i-th edge collapse operation are chosen in such a way that Eq. 7.15 is minimized:

$$\min\ (E(\boldsymbol{M^{n-i}}) - E(\boldsymbol{M^{n-i+1}})) \qquad \text{(EQ 7.15)}$$

The algorithm that constructs the PM representation described in this section only used the geometric information to compute the error associated with an edge collapse operation. If other information is available, such as normals, material properties, shading techniques, texture coordinates, then this could also be used in the evaluation of the error $E(\boldsymbol{M^i})$. If the representation would be used in a seismic ray tracer for example, the error associated with the normals at each triangle would be as important as the error associated with the geometry position of the vertices. The error norm should then try to find the edge collapse that minimizes both the changes in the geometry and in the normals.

A more detailed discussion on the progressive mesh representation can be found in the original papers of Hoppe [43].

7.3 Pair Contraction

In [37], Michael Garland and Paul S. Heckbert presented a surface simplification algorithm based on a quadratic error metric.

The algorithm proposed in this paper is similar in flavor to the progressive mesh scheme. The idea is to build a sequence of meshes $\boldsymbol{M^n}, \ldots, \boldsymbol{M^0}$ from an input mesh $\hat{\boldsymbol{M}} = \boldsymbol{M^n}$.

The two most noticeable differences between this representation and PM are:

- Instead of using an edge collapse operation (as in Section 7.2) Garland created a new operator called *pair contraction*. A pair contraction $(\boldsymbol{v_1}, \boldsymbol{v_2}) \rightarrow \bar{\boldsymbol{v}}$ introduces a new vertex $\bar{\boldsymbol{v}}$ in the mesh, moves the vertices $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ to $\bar{\boldsymbol{v}}$, connects all the incident edges of $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ to $\bar{\boldsymbol{v}}$, and finally removes $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ from the mesh.

  This operator is very similar to the edge collapse operator; the most important difference between the two operators is that the pair contraction operator can contract two vertices that do not share an edge, as shown in the right of Figure 7.6. The resulting representation has therefore the ability to change the topology of the mesh.

  From the definition given above any pair of vertices $(\boldsymbol{v_1}, \boldsymbol{v_2})$ could be used for a pair contraction. This would not generate high quality results in general, since merging distant vertices could introduce visual artifacts as well as self-intersections in the representation. The author introduced therefore the concept of a *valid contractions*:

Lemma 2: *a pair contraction of* $(\boldsymbol{v_1}, \boldsymbol{v_2})$ *is defined as valid if either an edge connects the two vertices or*

$$\|\boldsymbol{v_1} - \boldsymbol{v_2}\| \leq t \qquad \text{(EQ 7.16)}$$

where t is an mesh dependent threshold parameter.

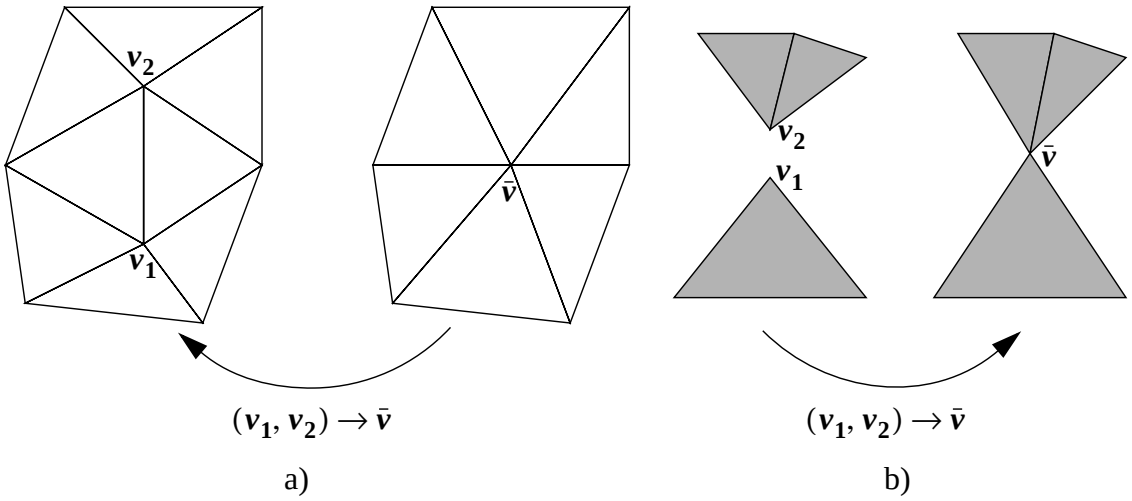


Figure 7.6 Pair contraction operator
a) The pair contraction operator collapses the edge (v_1, v_2)
b) The pair contraction operator collapses two vertices that do not share an edge

- The second important difference is how the error norm is used to evaluate the error introduced by a pair contraction operation.
  In PM the error is measured with an energy function: the lower the change in energy the better the edge collapse.
  Garland and Heckbert constructed a quadratic error metric: each vertex is assigned a 4×4 matrix Q, and the error at a vertex is defined as

$$\Delta(\boldsymbol{v}) = \boldsymbol{v}^T \cdot Q \cdot \boldsymbol{v} \qquad \text{(EQ 7.17)}$$

$$Q = \sum_{p \in \text{planes}(v)} K_p \qquad \text{(EQ 7.18)}$$

$$K_p = \boldsymbol{p} \cdot \boldsymbol{p}^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \qquad \text{(EQ 7.19)}$$

$$p = \begin{bmatrix} a & b & c & d \end{bmatrix} \qquad \text{(EQ 7.20)}$$

  Basically the matrix Q contains the information needed to compute the sum of the distances of the vertex $\boldsymbol{v}$ to a set of planes. This error metric allows to compute the position of the vertices that minimizes the error.

  The main advantages of quadratic error metrics are their efficiency, since the storage requirement is of 16 floating point values per vertex and the error can be evaluated with

two matrix-vector multiplications, and the intuitive understanding of what the error means.

Once these two components have been defined, the rest of the algorithm is very similar to PM: at iteration i the error introduced by every legal pair contractions is measured, and the contraction that introduces the smallest error is used to compute the mesh $\boldsymbol{M^{n-i}}$.

7.4 Progressive Compression of 3D Meshes

Li et al. [47] developed a strategy to compress 3D meshes progressively. The interesting aspect of this approach is that both the topology and the geometry are progressively encoded in the same bitstream. In this way it is possible to transmit the most important information first.

In order to build the bit stream, two operations must be available:

- *Coding of topological information*: this information is coded using a vertex removal strategy, such as the algorithm constructed by William Schroeder: unimportant vertices are removed, and the hole that is generated by the removal operation is re-triangulated using a divide-and-conquer strategy. An example of vertex removal is shown in Figure 7.4 in Section 7.1.

  This is a simple approach and it allows to remove vertices progressively. In order to reconstruct the original mesh, i.e. to invert the removal operation, it is necessary to store enough information to recover the neighborhood of the vertex that has been removed. Li uses the following information to invert a vertex removal operation:

  1. The index of one of the new triangles generated by the vertex removal operation.
  2. One bit per edge: a value of 1 specifies that the adjacent triangle was also generated by the removal operation, a 0 specifies the adjacent triangle wasn't generated.

  A vertex is then re-inserted in the mesh by first computing its star and then applying a trivial re-triangulation. The star is retrieved by finding all the triangles that were inserted during the vertex removal step: the index of one of these triangles is stored in the representation, and the others can be found using the 1's and 0's. An example is shown in Figure 7.7.

- *Coding of geometric information*: A second operator is needed to encode the geometric information. In his original paper Li used a progressive quantization approach.

  Standard quantization algorithms map a value $x \in S_1$, where S_1 is a set of finite size, onto a value $x' \in S_2$, where S_2 is a set of finite size and $|S_1| \gg |S_2|$. The standard example consists in mapping a float or double value, a 32 or 64 bit value, onto a smaller set of size 2^i, where i is a parameter that specifies the trade-off between the quality of the quantization and the compression rate. If this approach is applied to the geometric information of a mesh, then it is possible to compress the information. The problem

with this approach is that the information in the bitstream is not sorted in order of importance: the vertices are stored sequentially.

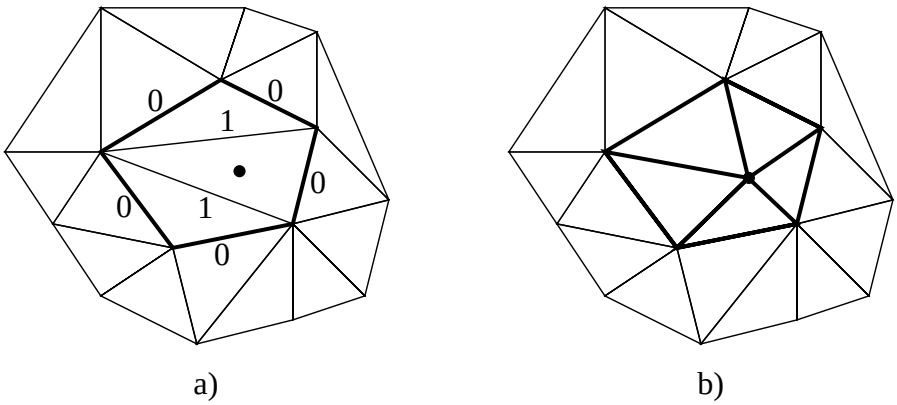


Figure 7.7 Introducing a vertex in the mesh
a) mesh before the insertion of the vertex.
b) mesh after the insertion of the vertex.

A better strategy consists in progressively quantizing the information from the most important part of the information to the least important. The most important information of the geometry component of a mesh corresponds to:

1. the most significant bits (MSB) of every vertex of the mesh.
2. the most important vertices in the mesh defined by the vertex removal algorithm.

Therefore, it makes sense to encode the MSB of the important vertices first, and only in a second step store the least significant bits (LSB) or the position of unimportant vertices.

This idea can be easily implemented using a threshold T_0 and the two rules defined in the following list:

0. *initialization*:

   T_0 = half of the maximum magnitude of the geometric data

1. *significance identification*:

   If $v_{j,c} > T_i$ then $S_{j,i,c} = 1$ and $E_{j,i,c} = v_{j,c} - 1.5 \cdot T_i$
   If $v_{j,c} < -T_i$ then $S_{j,i,c} = -1$ and $E_{j,i,c} = v_{j,c} + 1.5 \cdot T_i$
   Otherwise $S_{j,i,c} = 0$ and $E_{j,i,c} = v_{j,c}$

2. *refinement quantization*:

   If $E_{j,i-1,c} \geq 0$ then $R_{j,i,c} = 1$ and $E_{j,i,c} = E_{j,i-1,c} - T_{i,c}$
   If $E_{j,i-1,c} < 0$ then $R_{j,i,c} = -1$ and $E_{j,i,c} = E_{j,i-1,c} + T_{i,c}$

where $v_{j,c}$ represents the c-th component of the vertex $\boldsymbol{v_j}$, the $S_{i,j,c}$ store the sign and the magnitude of the c-th component of the vertex $\boldsymbol{v_j}$, the $E_{i,j,c}$ store the part of the

c-th component of $\boldsymbol{v_j}$ that has not been quantized yet, $R_{j,i,c}$ are the quantized bits of the c-th component of $\boldsymbol{v_j}$, and the T_i are the threshold values used in the process.

The significance identification rule is used for the first k iterations, where k is defined by $|v_{j,c}| < T_i, i = 0 \ldots k-1 \wedge |v_{j,c}| > T_k$, and once $S_{j,k,c}$ has been set to either 1 or -1 the refinement quantization rule will be used for the remaining iterations.

The geometric value of a vertex can then be reconstructed from the sequence $\boldsymbol{S_{j,0}}, \ldots, \boldsymbol{S_{j,k}}, \boldsymbol{R_{j,k+1}}, \ldots, \boldsymbol{R_{j,m}}$, where m corresponds to the number of bits used in the quantization process.

Finally, the values $\boldsymbol{S_{j,i}}$ and $\boldsymbol{R_{j,l}}$ can be encoded progressively by first encoding all the values at iteration 0, then the values of iteration 1 and so on.

The two algorithms presented in this section, *vertex removal* and *progressive quantization*, allow us to simplify the mesh by reducing the number of vertices and to store the geometric information according to its importance. The next step consists in merging the two algorithms together, in order to build a single bitstream that encodes both geometric and topological information. One possible coding strategy consists of:

1. Set T to the value of the maximum magnitude of all prediction residues of every vertex that has to be removed, then define a threshold sequence for the geometric information

$$T_0 = \frac{T}{2}, T_{i+1} = \frac{T_i}{2} \qquad \text{(EQ 7.21)}$$

2. Define a threshold sequence for the topologic information

$$\infty = S_{-1} > S_0 > S_1 > \ldots \qquad \text{(EQ 7.22)}$$

   where $S_i = \sqrt{\frac{K}{6}} T_i$, and K represents the average number of bits required for a vertex addition.

3. In layer i all the vertices with prediction residue $[S_{i-1}, S_i]$ are added, and the geometric information of all the vertices present in the mesh are refined up to T_i.

7.5 Evaluation

In this section the mesh based methods described in Section 7.1 through Section 7.4 will be evaluated using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: The main strength of mesh based representations is their ability to model most triangular meshes. All of the representations described in this section can model two-manifolds with boundaries, and most of them can model non-manifold surfaces as well.

- *Computation of surface-surface intersections*: The previous representations described in Section 4 through Section 6 could in theory use two different approaches to compute intersections, using either a separate bounding box data structure or algebraic methods. Mesh based representations can only use a hierarchical bounding box data structure to accelerate the intersection computation, since algebraic methods require some degree of regularity in the surface and are usually applied to higher order functions.

  Most of the mesh based representations construct a multiresolution representation which is not described by a tree, but by a direct acyclic graph (DAG). As a consequence more triangles need to be tested during the computation of the intersection, since there are many more dependencies in the representation.

- *Scalable representation*: The mesh based representation described in this section are all scalable, since they all fulfill the following properties:

  - The representations are compact: the progressive mesh representation and Heckbert and Garland representation based on pair contraction require less storage than a trivial single resolution representation of a mesh.

    Li brings this concept even further and describes a method to reduce the storage requirements even more by encoding both the topological and geometric information in a bitstream.

    Schroeder's algorithm is not as advanced, and the storage requirements are therefore higher. The problem of this representation is that it is difficult to store the topology at all the resolutions in a compact way, since the connectivity changes after each vertex removal.

  - The representations are capable of generating simplified approximations of a mesh, thus allowing us to interact with very complicated and dense geological models. These approximations are usually not "optimal", i.e. they do not minimize an error norm, they are just good approximations constructed using greedy algorithms. The approximations are constructed starting from a coarse representation of the surface and adding enough details to meet the needs of the user: for example using the progressive mesh representation the approximation would be constructed by first taking the coarse representation and by applying n vertex split operations.

    Schroeder's representation based on vertex removal is the only exception. Surfaces are not stored in a hierarchical representation; the algorithm is only capable to take a fine resolution mesh and simplify it.

  - The construction of the representation is also very efficient: all the algorithms presented in this section can construct a representation in $O(n \cdot \log n)$ time. If the user specifies more complex error norms the total complexity of the algorithms can increase, usually resulting in $O(n^2)$ algorithms.

    Once the representation of a mesh has been constructed these algorithms can build an approximation of the full resolution mesh in linear time.

- *Modeling of non-manifold singularities, tears and cracks*: Basically all the mesh based representations described in this section can handle non-manifold surfaces.

Schroeder's representation can model two-manifolds easily, since special masks are defined in order to handle these singularities (see Figure 7.1). Pseudo-manifolds are not handled explicitly in the original paper, but it would be possible to extend the algorithm to correctly handle these singularities.

The original paper on progressive mesh did not address the problem of representing non-manifolds using the edge collapse and vertex split operators, but in the paper of progressive simplicial complexes J. Popovic and H. Hoppe [67] extend the theory of the progressive mesh to non-manifolds. Pseudo-manifolds were not handled explicitly, but it should be possible to extend the representation to handle these singularities as well.

The representation constructed by Heckbert and Garland does handle non-manifolds as well, since the representation is very similar to the progressive mesh. Pseudo-manifolds are not handled explicitly, but it should be possible to extend the representation to handle them correctly.

Li's representation is based on Schroeder's representation, and therefore it can handle non-manifold singularities well, and it could be extended to handle pseudo-singularities.

The mesh based representations can model tears and cracks as piecewise linear curves embedded in the piecewise linear representation of the surface. The data structure that describes the curves is separate from the multiresolution representation of the mesh.

- *Error modeling*: During the construction of the representations the mesh based algorithms minimize some error norm. The resulting representations are therefore usually capable to construct approximations which satisfy some error conditions.
  All the algorithms presented in this section have the drawback of constructing the surface representations using a greedy approach: Schroeder and Li remove the vertices that introduce the smallest error, and Hoppe and Heckbert-Garland collapse pairs of vertices that introduce the smallest error. As a consequence the representations will usually not be optimal, i.e. they use too many triangles to guarantee any error norm.
- *Smoothness of the surface*: All mesh based representations assume the surface is piecewise linear, and therefore they cannot build smooth representations of surfaces.
- *Multiresolution editing*: Schroeder's representation does not support multiresolution editing since it does not construct a multiresolution representation of surfaces.

  Hoppe's and Heckbert-Garland's representations construct a multiresolution representation of surfaces, so it may be argued that it is possible to build a multiresolution editing tool on top of the representation. A multiresolution editing tool built on top of these representations would not generate good edits for two reasons:

  1. an edit would affect both low and high frequency components of the mesh. Good edit operations are usually only performed on the low-pass component of the mesh which contains the structural information the user wants to edit.
  2. the information needed to reconstruct the full resolution mesh is not stored using details, nor it is encoded using local frames. As a result an edit at any level of resolution but the finest would result in meaningless changes in the mesh.

  Li's representation does not construct a multiresolution representation explicitly, but the bitstream it generates implicitly defines a multiresolution representation that could

be used to construct a multiresolution editing tool. Such a tool would have the same two problems described above for Hoppe's and Heckbert-Garland's representations.

- *Surface fitting*: There is no fitting algorithm that takes as input a collection of points and generates a representation based on Schroeder's algorithm. However it could be possible to construct a triangular mesh from an unorganized cloud of points and then compute Schroeder's representation as a post-process.

  In his thesis Hoppe constructed a fitting scheme which constructed a representation very similar to his progressive mesh representation. This algorithm could be adapted to generate PM representations automatically.

  Since the PM representation is very similar to the representation constructed by Heckbert and Garland it would be possible to take Hoppe's fitting scheme and adapt it to use the quadratic error norm and to automatically construct a representation based on the pair contraction operator.

  Li's representation is based heavily on Schroeder's algorithm, so the same comment applies to this scheme: the simplest approach would be to construct a triangular mesh that fits the collection of points and in a post-process to construct Li's representation.

- *Support of local high variation in the curvature of the surface*: All the mesh based representation can model local high variation in the curvature since the underlaying surface representation is a piecewise linear surface.

- *Changes of the surface over time*: Depending on the changes in the topology of the mesh over time, these changes can be model more or less easily. The same classification of the changes used in Section 4.5 for the pseudo-wavelets applies here as well:

  1. If the connectivity of the mesh remains the same at all times t_i then the changes in the geometry can be encoded easily by storing a function $f(t)$ at each vertex instead of a single vector value (x, y, z).
  2. If the connectivity changes, but the topological type remains the same at all times t_i then changes can be encoded by first computing a remesh of the surface at all the times t_i, and then by applying the solution constructed in 1.
  3. If the topological type of the surface changes as well in time, then the only solution is to store different representations of the surface at each time t_i

8 Signal Processing for Meshes

In this last section some of the representations based on signal processing tools will be analyzed. These primitives are used to smooth the surface usually by minimizing the Laplacian operator. There are four main contributions to this field:

- Gabriel Taubin [86] was the first that tried to map standard filtering techniques to meshes with arbitrary connectivity.
- Leif Kobbelt [48] used the work of Taubin as a starting point to construct a multiresolution editing tool for meshes with arbitrary connectivity. In order to accomplish this he constructed an operator to smooth surfaces as well as a local frame operator.
- Igor Guskov et al. constructed in [46] a better smoothing operator, and used it to build non-uniform subdivision schemes and a multiresolution representation of meshes.
- Desbrun et al. constructed in [24] a smoothing operator based on the notion of curvature flow, and used implicit integration instead of the less stable explicit integration to accelerate the convergence of the algorithm.

8.1 Signal Processing on Meshes

Gabriel Taubin presented in [86] a new idea based on signal processing techniques that allows to smooth surfaces while imposing different types of constraints. In this section the basic ideas of this signal processing approach will be discussed briefly.

8.1.1 Fairing of One Dimensional Curves

Given a curve defined by the vector $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})^T$, where the values $\boldsymbol{x_i}$, $i = 1, \ldots, n$ represent the vertices and the edges are defined between each pair of vertices $(\boldsymbol{x_i}, \boldsymbol{x_{i+1}})$, $i = 1, \ldots, n-1$, the Laplacian operator is defined as

$$\Delta \boldsymbol{x_i} = \frac{1}{2} \cdot (\boldsymbol{x_{i-1}} - \boldsymbol{x_i}) + \frac{1}{2} \cdot (\boldsymbol{x_{i+1}} - \boldsymbol{x_i}) \qquad \text{(EQ 8.1)}$$

and can be represented in matrix form as

$$\Delta \boldsymbol{x} = -K \cdot \boldsymbol{x} \qquad \text{(EQ 8.2)}$$

The one dimensional curve can now be faired with two different methods:

1. One obvious strategy consists in computing the Discrete Fourier Transform (DFT) of the signal $\boldsymbol{x}$ and removing the high energy component from the signal. Using a Fast Fourier Transform (FFT) this could be accomplished in $O(n \cdot \log n)$

   Instead of computing the FFT of the input signal $\boldsymbol{x}$ it is also possible to compute the eigenvalues k_i, $i = 1, \ldots, n$ and the eigenvectors $\boldsymbol{u_i}$, $i = 1, \ldots, n$ of the matrix K. The vector $\boldsymbol{x}$ could then be re-written as

$$\boldsymbol{x} = \sum_{i=1}^{n} \xi_i \cdot \boldsymbol{u}_i \qquad \text{(EQ 8.3)}$$

$$\xi_i = \langle \boldsymbol{x}, \boldsymbol{u}_i \rangle \qquad \text{(EQ 8.4)}$$

The high frequencies in the signal could be eliminated by leaving out the eigenvectors that correspond to higher frequencies, which are defined as the eigenvector whose corresponding eigenvalue is larger than a threshold. This second strategy has the same complexity as the FFT approach: $O(n \cdot \log n)$.

Note, that the removal of high frequency information from the DFT of the signal $\boldsymbol{x}$ results in the same faired signal as the reconstruction of the signal $\boldsymbol{x}$ via Eq. 8.3 using only eigenvectors that correspond to low frequency information.

An example of surface fairing using this strategy is presented in Figure 8.1. The images were obtained using Eq. 8.3 to reconstruct the signal:

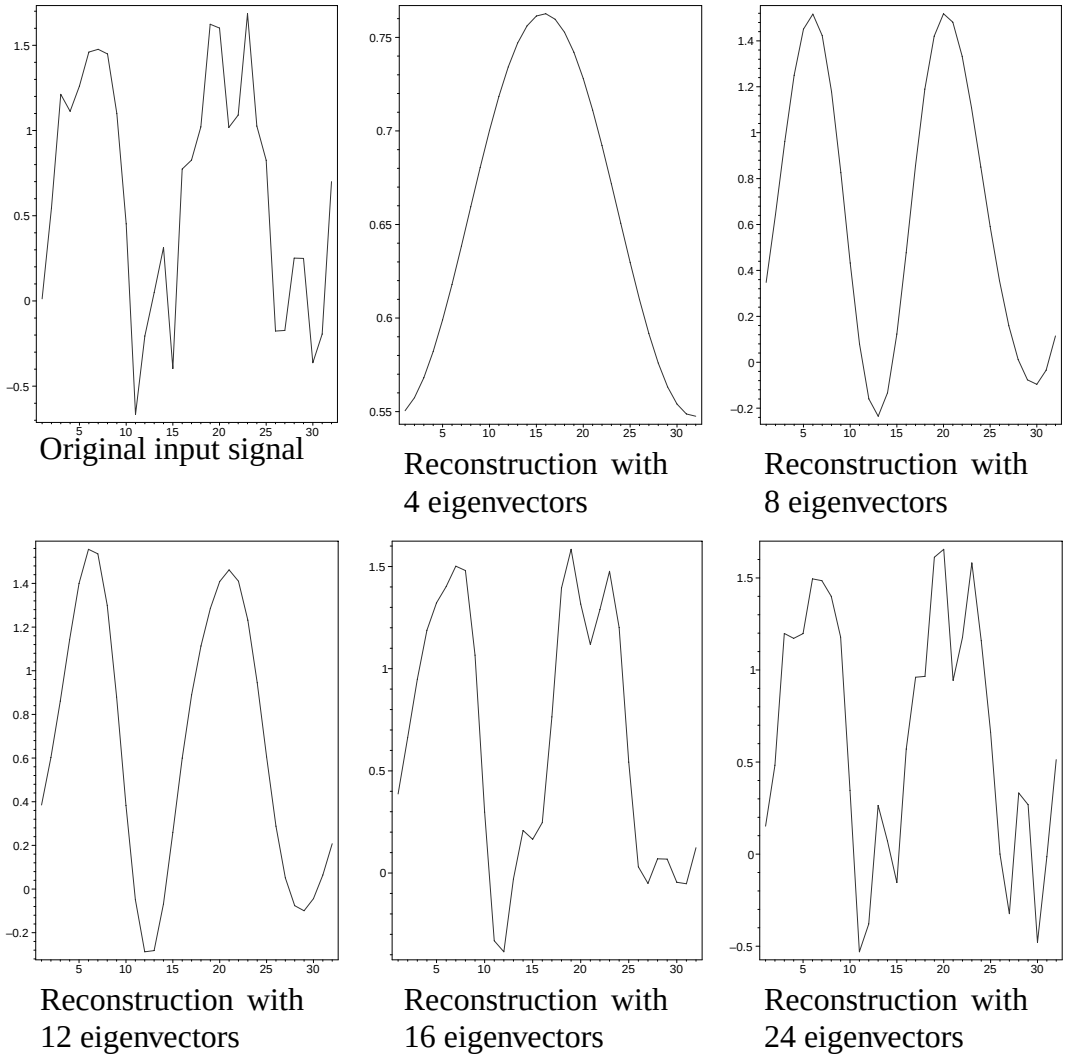


Figure 8.1 Fairing of a 1D signal using an eigenanalysis

2. A simpler and faster operator that allows to remove high energy information from a one dimensional curve is the Gaussian filter, or more generally any low-pass filter. These filters do not compute the exact projection of a signal into the space of the low-frequency signals, but they compute good approximations very fast.

   One of the most simple filter is the Gauss filter, which can be defined as

$$\boldsymbol{x}' = \boldsymbol{x} + \lambda \cdot \Delta \boldsymbol{x} \qquad \text{(EQ 8.5)}$$

   with $0 < \lambda < 1$.

   An example of this low-pass filter is given in Figure 8.2:

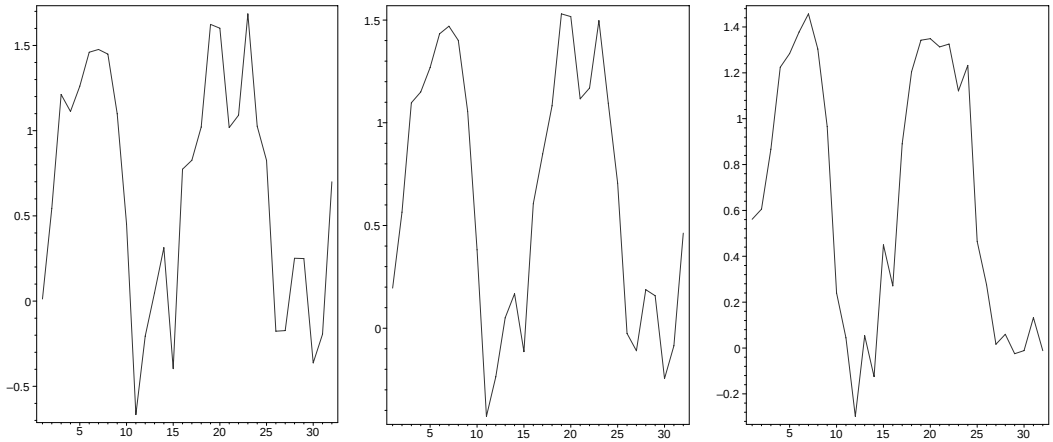


Figure 8.2 Gaussian filter

   The most important drawback of the gaussian filter is that while it smooths the signal it also shrinks it, and thus the area or volume of the signal is not preserved in any way. The method constructed by Taubin, which will be presented in the next section, will overcome this limitation.

8.1.2 Fairing of Arbitrary Meshes

The strategies presented in the previous section allow to smooth a one dimensional signals. In this section a smoothing operator will be described that can be applied on two-manifold meshes with arbitrary connectivity.

The Laplacian operator at the vertex $\boldsymbol{x_i}$ is defined as the weighted average of the edge vectors $(\boldsymbol{x_i} - \boldsymbol{x_j})$:

$$\Delta \boldsymbol{x_i} = \sum_{j \in \text{star}(\boldsymbol{x_i})} w_{i,j} \cdot (\boldsymbol{x_i} - \boldsymbol{x_j}) \quad \text{(EQ 8.6)}$$

$$\sum_{j \in \text{star}(\boldsymbol{x_i})} w_{i,j} = 1 \quad \text{(EQ 8.7)}$$

The matrix K can be defined as

$$K = I - W \quad \text{(EQ 8.8)}$$

where I is the identity matrix, and W is the matrix whose (i, j) element corresponds to $w_{i,j}$.

The low-pass filtering can now be redefined in matrix form as

$$\boldsymbol{x}^{(k+1)} = f(K) \cdot \boldsymbol{x}^{(k)} \quad \text{(EQ 8.9)}$$

The goal is now to find a function $f(K)$, for which $f(K)^N \approx 1$ for the low frequencies, and $f(K)^N \approx 0$ for the high frequencies present in the surface. A simple function satisfying this criteria is

$$f(K) = (I - \mu \cdot K) \cdot (I - \lambda \cdot K) \quad \text{(EQ 8.10)}$$

where $\lambda > 0$ and $\mu < -\lambda$.

The fairing operator has been applied to a synthetic model that was generated using the Loop subdivision scheme (see Section 5.1.3) and successively noising the right half of the mesh. The results are presented in Figure 8.3:

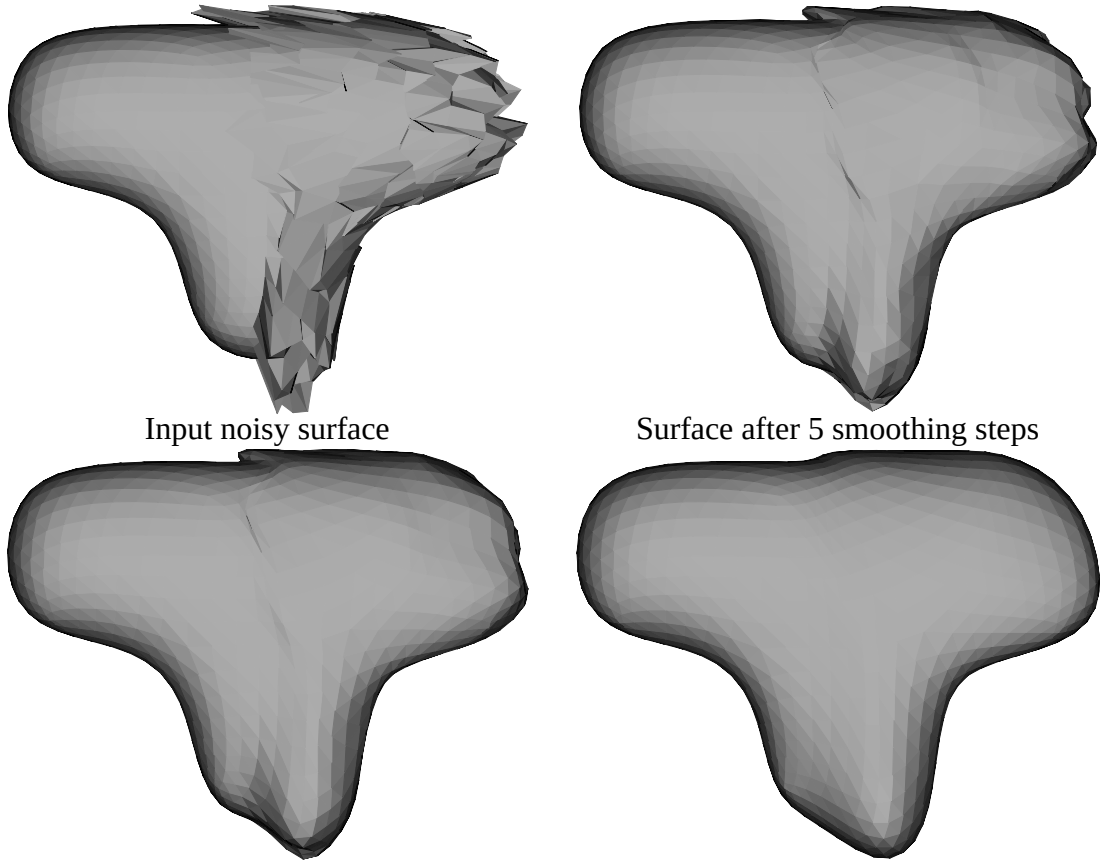


Figure 8.3 Non-shrinking fairing strategy

This fairing algorithm does not suffer the problem of shrinkage as the simple Gaussian filter does described in Section 8.1.1, since after the Gaussian smoothing step represented by $(I - \lambda \cdot K)$ in Eq. 8.10 an un-shrinking operation represented by $(I - \mu \cdot K)$ in Eq. 8.10 is performed.

This algorithm can be further generalized by imposing constraints to the fairing process. The different constraints that can be applied are quickly described below:

- *Interpolatory constraints*: fix the position of some vertices so that they are not moved during the smoothing step. There are two possible implementations of these constraints, one resulting in a non-smooth neighborhood of the interpolated vertices, the other in a smooth neighborhood.
- *Smooth deformations*: the smooth interpolatory constraint can also be used to build smooth deformations of the mesh.
- *Hierarchical constraints*: labelling the vertices using some pre-defined rules makes it possible to smooth specific regions of the mesh; for example it is possible to smooth

only the boundary of the mesh or to define different regions of the mesh that will be smoothed separately, which means that there will be no smoothing across the boundaries of the regions.

- *Tangent plane constraints*: by defining the normal at a vertex $\boldsymbol{v}$ it is possible to put constraints on it.

Remark: the fairing operator presented by Taubin can also be used to construct a subdivision scheme that does not suffer the problem of shrinkage, as opposed to the classic subdivisions presented in Section 5.1.1 through Section 5.1.3. This is accomplished by using the fairing operator that was developed in this section to smooth the geometry in conjunction with any subdivision operator. This subdivision scheme was used in [90] to construct a multiresolution editing tool based on subdivision surfaces.

Figure 8.4 shows the subdivision surface generated using the fairing operator to smooth the geometry of the mesh:

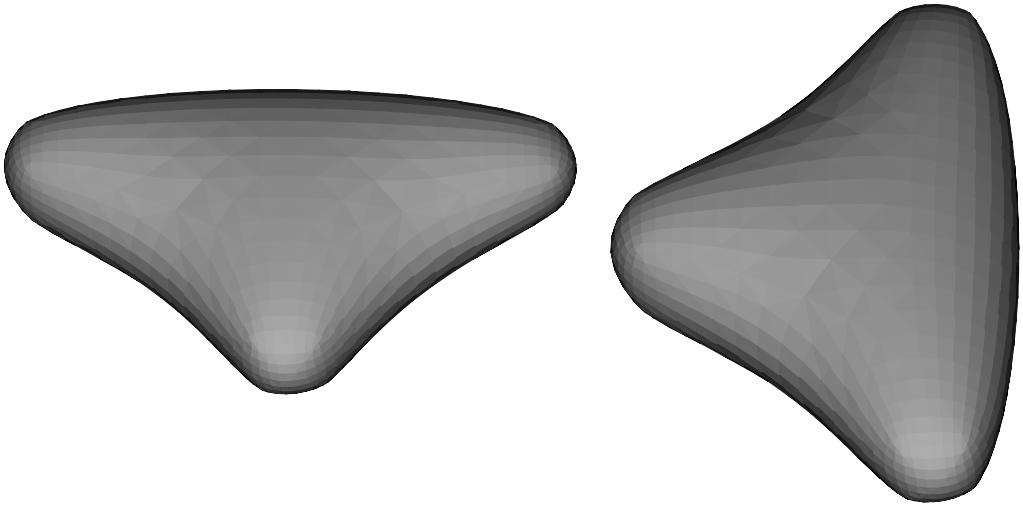


Figure 8.4 Non-shrinking subdivision surface using the fairing scheme

8.2 Multiresolution Modeling of Meshes

Leif Kobbelt et al. introduced in [48] a multiresolution editing tool that can handle meshes with arbitrary connectivity. The authors achieve this goal by defining two operators on the mesh: a *local frame* operator and a *discrete fairing* operator.

8.2.1 The Local Frame Operator

Local frames were first introduced in [36] and they are used to encode the geometric information of vertices in a local coordinate system. Since vertices are encoded using a local coordinate system, edit operations at any level of resolution are automatically propagated to the full resolution mesh.

Local frames could be defined in many different ways. In the paper of Kobbelt they are constructed using the following strategy:

1. Construct a *local frame space* that will be used to code the vertex. This space is defined as a parametric function

$$F(u, v) = \boldsymbol{a} + u \cdot \boldsymbol{a_u} + v \cdot \boldsymbol{a_v} + \frac{u^2}{2} \cdot \boldsymbol{a_{uu}} + u \cdot v \cdot \boldsymbol{a_{uv}} + \frac{v^2}{2} \cdot \boldsymbol{a_{vv}} \qquad \text{(EQ 8.11)}$$

over a local neighborhood of the mesh, as shown on the left of Figure 8.5:

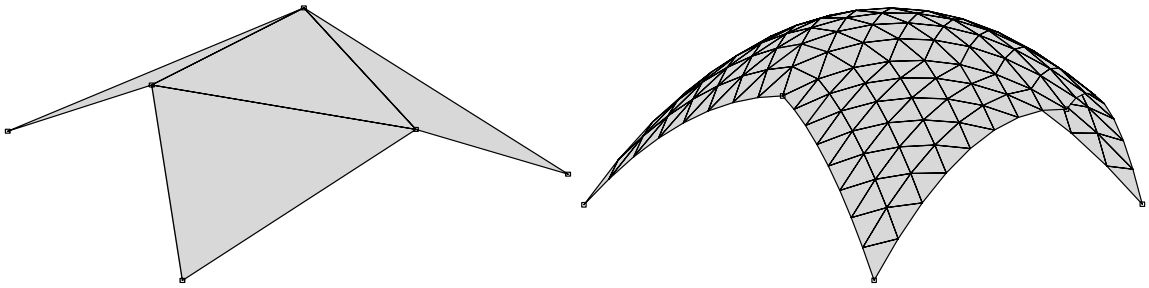


Figure 8.5 Local frame space define over a local neighborhood of a triangle mesh

In order to evaluate the unknown $\boldsymbol{a}$, $\boldsymbol{a_u}$, $\boldsymbol{a_v}$, $\boldsymbol{a_{uu}}$, $\boldsymbol{a_{uv}}$, and $\boldsymbol{a_{vv}}$ in Eq. 8.11 it is first necessary to define the support of the frame space, which is defined by a triangle and the three neighbor triangles, as shown in Figure 8.6:

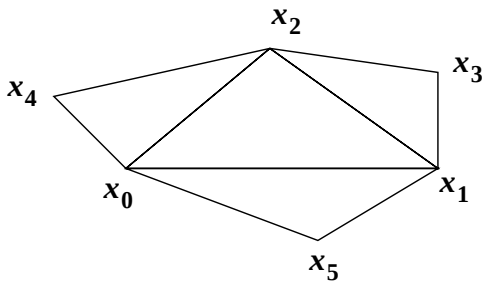


Figure 8.6 Support of the frame space

Next, a parametrization (u_i, v_i) for the points $\boldsymbol{x_i}, i = 0, \ldots, 5$ must be constructed. The parameter position of the first three vertices can be freely set to

$$(u_0, v_0) = (0, 0) \qquad \text{(EQ 8.12)}$$

$$(u_1, v_1) = (1, 0) \qquad \text{(EQ 8.13)}$$

$$(u_2, v_2) = (0, 1) \qquad \text{(EQ 8.14)}$$

The parameter position of $\boldsymbol{x_3}, \boldsymbol{x_4}, \boldsymbol{x_5}$ is not unique, and it depends on the projection operator.
A simple idea is to first project these points on the plane defined by $(\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2})$:

$$\boldsymbol{x_i}' = \boldsymbol{x_i} - (\boldsymbol{x_i}^T \cdot \boldsymbol{n}) \cdot \boldsymbol{n} \quad , i = 3, \ldots, 5 \qquad \text{(EQ 8.15)}$$

$$\boldsymbol{n} = \frac{\boldsymbol{d_1} \times \boldsymbol{d_2}}{\left\| \boldsymbol{d_1} \times \boldsymbol{d_2} \right\|} \tag{EQ 8.16}$$

$$\boldsymbol{d_1} = \frac{\boldsymbol{x_1} - \boldsymbol{x_0}}{\left\| \boldsymbol{x_1} - \boldsymbol{x_0} \right\|} \tag{EQ 8.17}$$

$$\boldsymbol{d_2} = \frac{\boldsymbol{x_2} - \boldsymbol{x_0}}{\left\| \boldsymbol{x_2} - \boldsymbol{x_0} \right\|} \tag{EQ 8.18}$$

Next, Eq. 8.12 - Eq. 8.18 can be used to compute the parametrization of the vertices $\boldsymbol{x_3}, \boldsymbol{x_4}, \boldsymbol{x_5}$:

$$(u_i, v_i) = (c_1, c_2) \tag{EQ 8.19}$$

where

$$\boldsymbol{x_i}' = c_1 \cdot \boldsymbol{d_1} + c_2 \cdot \boldsymbol{d_2} \tag{EQ 8.20}$$

and

$$c_1 = \frac{\boldsymbol{d_1}^T \cdot \boldsymbol{x_i} - (\boldsymbol{d_1}^T \cdot \boldsymbol{d_2}) \cdot (\boldsymbol{d_2}^T \cdot \boldsymbol{x_i})}{\left\| \boldsymbol{d_1} - (\boldsymbol{d_1}^T \cdot \boldsymbol{d_2}) \cdot \boldsymbol{d_2} \right\|^2} \tag{EQ 8.21}$$

$$c_2 = \boldsymbol{d_2}^T \cdot \boldsymbol{x_i} - (\boldsymbol{d_1}^T \cdot \boldsymbol{d_2}) \cdot c_1 \tag{EQ 8.22}$$

Once the parametrization of the vertices $\boldsymbol{x_0}, \ldots, \boldsymbol{x_5}$ has been computed it is possible to compute the unknown in Eq. 8.11:

$$\boldsymbol{a} = \boldsymbol{x_0} \tag{EQ 8.23}$$

$$\boldsymbol{a_u} = (\boldsymbol{x_1} - \boldsymbol{x_0}) - \frac{1}{2} \cdot \boldsymbol{a_{uu}} \tag{EQ 8.24}$$

$$\boldsymbol{a_v} = (\boldsymbol{x_2} - \boldsymbol{x_0}) - \frac{1}{2} \cdot \boldsymbol{a_{vv}} \tag{EQ 8.25}$$

$$\begin{bmatrix} \frac{1}{2}u_3(u_3-1) & u_3v_3 & \frac{1}{2}v_3(v_3-1) \\ \frac{1}{2}u_4(u_4-1) & u_4v_4 & \frac{1}{2}v_4(v_4-1) \\ \frac{1}{2}u_5(u_5-1) & u_5v_5 & \frac{1}{2}v_5(v_5-1) \\ \tau & 0 & 0 \\ 0 & 2\tau & 0 \\ 0 & 0 & \tau \end{bmatrix} \begin{bmatrix} \boldsymbol{a_{uu}} \\ \boldsymbol{a_{uv}} \\ \boldsymbol{a_{vv}} \end{bmatrix} = \begin{bmatrix} (\boldsymbol{x_3} - \boldsymbol{x_0}) + u_3(\boldsymbol{x_0} - \boldsymbol{x_1}) + v_3(\boldsymbol{x_0} - \boldsymbol{x_2}) \\ (\boldsymbol{x_4} - \boldsymbol{x_0}) + u_4(\boldsymbol{x_0} - \boldsymbol{x_1}) + v_4(\boldsymbol{x_0} - \boldsymbol{x_2}) \\ (\boldsymbol{x_5} - \boldsymbol{x_0}) + u_5(\boldsymbol{x_0} - \boldsymbol{x_1}) + v_5(\boldsymbol{x_0} - \boldsymbol{x_2}) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{EQ 8.26}$$

The parameter $\tau \in [0, 1]$ in Eq. 8.26 is used to control the curvature of the frame space. If τ is set to zero the curvature is not constrained, and the local frame space $F(u, v)$ will interpolate the six points $\boldsymbol{x_i}$, $i = 0, \ldots, 5$ that generate it.

2. In a second step a vertex $\boldsymbol{q}$ has to be coded in the local frame space. The vertex is coded as a triple $(\hat{u}, \hat{v}, h)$, where $(\hat{u}, \hat{v})$ is the parameter position of the point on $F(u, v)$ whose normal intersects the vertex $\boldsymbol{q}$, and h specifies the distance between the frame space and the vertex $\boldsymbol{q}$.

   The triple $(\hat{u}, \hat{v}, h)$ can be computed by first making an initial guess on $(\hat{u}, \hat{v})$, for example

$$(u, v) = \left(\frac{1}{3}, \frac{1}{3}\right) \tag{EQ 8.27}$$

   and then using an iterative algorithm to relax the approximation to the solution. In the original paper a Newton iteration was used, as shown in Eq. 8.28 through Eq. 8.30

$$(u, v) \leftarrow (u, v) + (\Delta u, \Delta v) \tag{EQ 8.28}$$

$$\boldsymbol{d} = \boldsymbol{q} - F(u, v) \tag{EQ 8.29}$$

$$\begin{bmatrix} F_u^T \cdot F_u & F_u^T \cdot F_v \\ F_u^T \cdot F_u & F_v^T \cdot F_v \end{bmatrix} \cdot \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} F_u^T \cdot \boldsymbol{d} \\ F_u^T \cdot \boldsymbol{d} \end{bmatrix} \tag{EQ 8.30}$$

   The system iterates until the approximation is close enough to the solution. Finally, the vector $\boldsymbol{d}$ is coded in a single value h:

$$h = \text{sign}\,(\boldsymbol{d}^T \cdot (F_u \times F_v)) \cdot \|\boldsymbol{d}\| \tag{EQ 8.31}$$

   It is possible to code $\boldsymbol{d}$ into a single value h, since $\boldsymbol{d}$ is parallel to the normal of the frame space $F(u, v)$ at the position $(\hat{u}, \hat{v})$.

An example of Newton iteration is given in Figure 8.7:

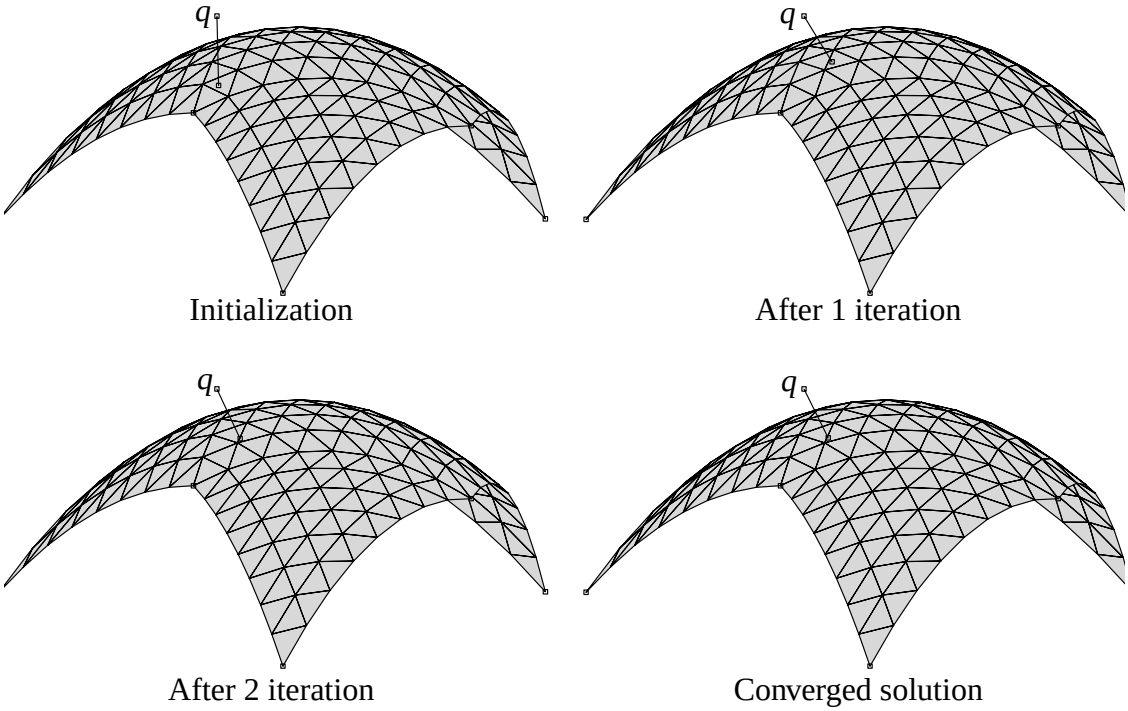


Figure 8.7 Newton iteration is used to compute local frame coordinates

The local frames can now be used with any existing decimation scheme to efficiently code vertices that have been removed in a local coordinate system. They can also be used to store the original position of the vertices that are being smoothed with the discrete fairing operator that will be introduced in the next subsection.

8.2.2 The Discrete Fairing Operator

The second operator that is needed to correctly edit meshes with arbitrary connectivity at multiple levels of resolution is a *discrete fairing* operator. This operator is used to smooth the mesh in order to remove high frequency information. The smoothing operator is very useful, since edit operations are usually applied only to the low-pass component of the mesh.

The discrete fairing has been implemented with the so-called *umbrella algorithm*. The goal of this algorithm is to minimize either the so-called *membrane energy* defined as

$$E_M(f) = \int f_u^2 + f_v^2 \qquad \text{(EQ 8.32)}$$

or the *thin plate energy* defined as

$$E_{TP}(f) = \int f_{uu}^2 + 2f_{uv}^2 + f_{vv}^2 \qquad \text{(EQ 8.33)}$$

Eq. 8.34 and Eq. 8.35 shows the partial differential equations (PDE) that correspond to the energy minimization of Eq. 8.32 and Eq. 8.33:

$$\Delta f = f_{uu} + f_{vv} = 0 \qquad \text{(EQ 8.34)}$$

$$\Delta^2 f = f_{uuuu} + 2 f_{uuvv} + f_{vvvv} = 0 \qquad \text{(EQ 8.35)}$$

The operators used to minimize both energies are local: the function that updates a vertex $\boldsymbol{x}$ uses only the information on the vertices surrounding $\boldsymbol{x}$, represented by the vertices $\boldsymbol{x_0}, \ldots, \boldsymbol{x_{n-1}}$ in Figure 8.8:

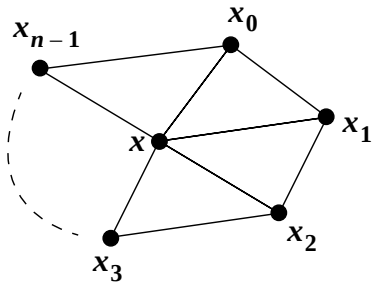


Figure 8.8 Vertices used in the umbrella algorithm

The parametrization of these vertices has a direct impact on the fairing function. Kobbelt has chosen a simple symmetric parametrization for the vertices $\boldsymbol{x_i}$, $i = 0, \ldots, n-1$:

$$(u_i, v_i) = \left(\cos\left(2\pi\frac{i}{n}\right), \sin\left(2\pi\frac{i}{n}\right)\right), i = 0, \ldots, n-1 \qquad \text{(EQ 8.36)}$$

The use of this parametrization allows to construct a discrete operator that models the Laplacian Δf

$$U(\boldsymbol{x}) = \frac{1}{n} \cdot \sum_{i=0}^{n-1} \boldsymbol{x_i} - \boldsymbol{x} \qquad \text{(EQ 8.37)}$$

as well as a discrete operator that models $\Delta^2 f$

$$U^2(\boldsymbol{x}) = \frac{1}{n} \cdot \sum_{i=0}^{n-1} U(\boldsymbol{x_i}) - U(\boldsymbol{x}) \qquad \text{(EQ 8.38)}$$

Finally, the fairing is obtained using a fixed-point iteration of either Eq. 8.37 or Eq. 8.38:

1. the solution of the discrete problem $U(\boldsymbol{x}) = 0$, which minimizes the membrane energy, is computed by using a simple fixed-point iteration:

$$\boldsymbol{x_i}^{(k+1)} = \boldsymbol{x_i}^{(k)} + U(\boldsymbol{x_i}^{(k)}) \qquad \text{(EQ 8.39)}$$

The algorithm iterates, until the system converges towards a mesh with the property $U(\boldsymbol{x}) = 0$ for all vertices $\boldsymbol{x}$.

An example of the relaxation procedure is shown in Figure 8.9:

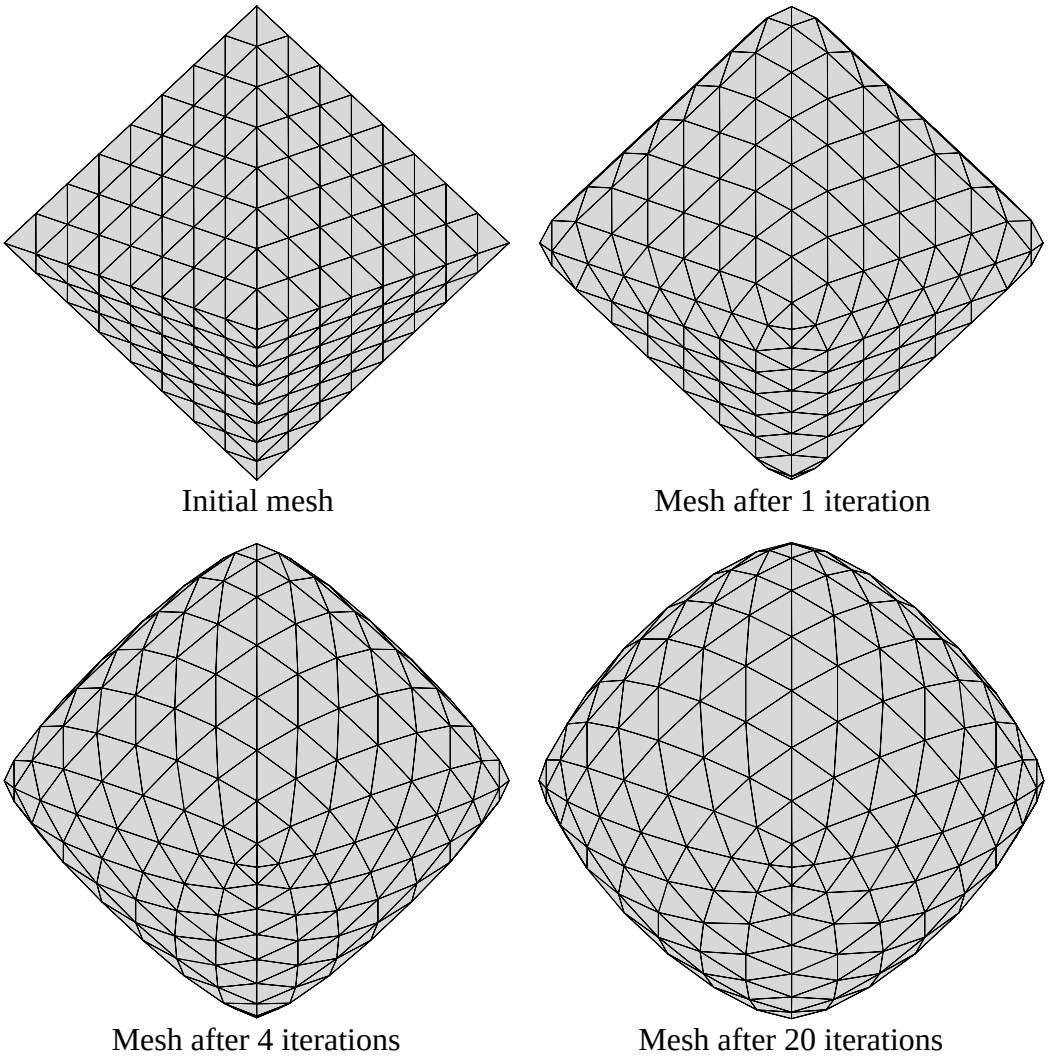


Figure 8.9 Minimization of the membrane energy

2. The solution of the discrete problem $U^2(\boldsymbol{x}) = 0$, which minimizes the thin plate energy, is slightly more complex, but it can also be solved with a fixed point iteration of the form

$$\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} - \frac{1}{\nu} \cdot U^2(\boldsymbol{x}_i^{(k)}) \qquad \text{(EQ 8.40)}$$

where

$$\nu = 1 + \frac{1}{\nu_i} \cdot \sum_{j=0}^{n-1} \frac{1}{\nu_{i,j}} \qquad \text{(EQ 8.41)}$$

The parameter ν_i represents the valence of the vertex $\boldsymbol{x_i}$, and $\nu_{i,j}$ represents the valence of the j-th neighbor vertex of $\boldsymbol{x_i}$. The algorithm will iterate Eq. 8.40 until a mesh with the property $U^2(\boldsymbol{x}) = 0$ is created.

An example of the iterative process that minimizes the thin plate energy is given in Figure 8.10:

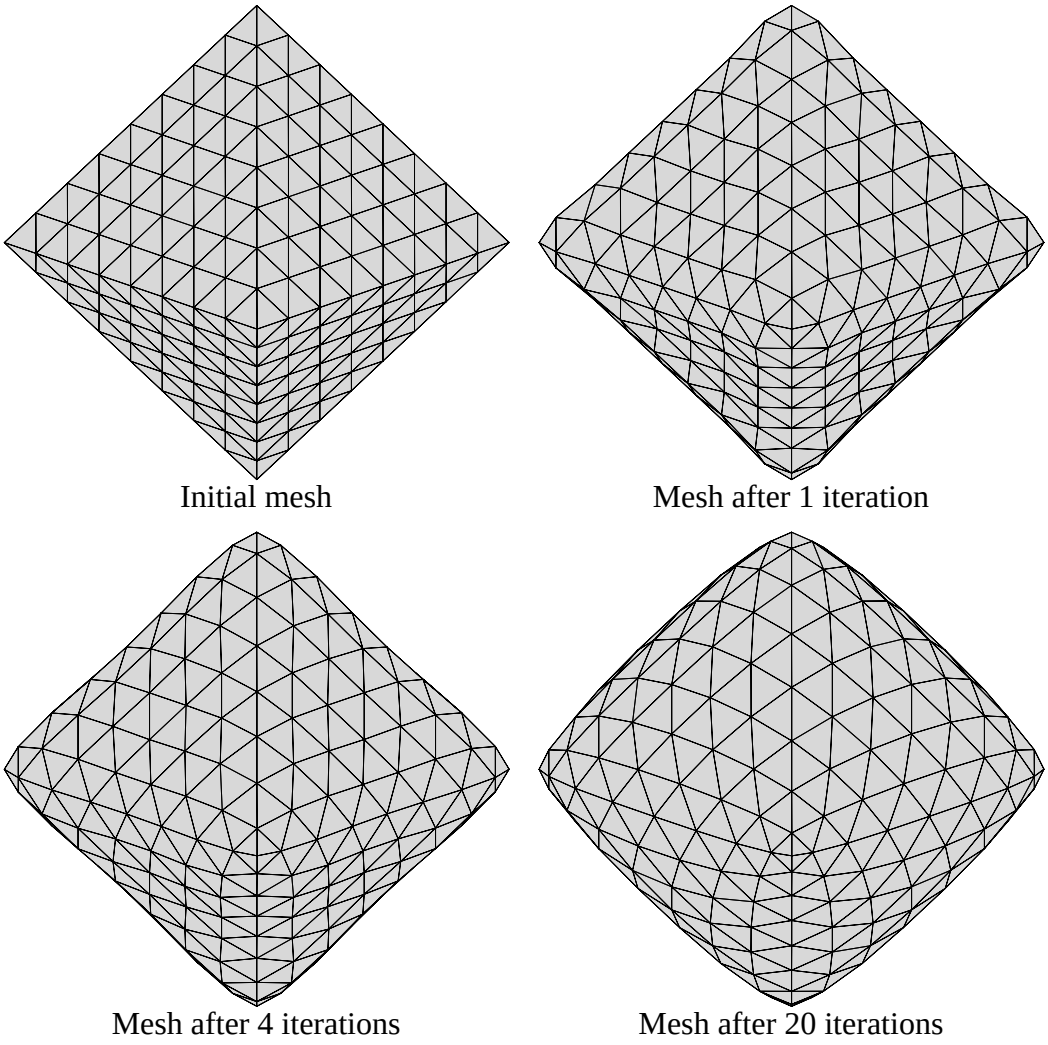


Figure 8.10 Minimization of the thin plate energy

8.2.3 Construction of an Edit Operator

The local frames and the discrete fairing operators developed in the previous sections can now be used to construct a multi-resolution editing tool that can handle meshes with arbitrary micro-topology. In this section the algorithm will be presented briefly; further information can be found in the original paper of Kobbelt:

- the first step consists in the coarsification of the mesh using any decimation algorithm, for example any of the algorithms presented in Section 7.1 through Section 7.4 could

be used. The position of the vertices removed from the mesh is stored in local frames, so that the edit operations automatically affect these vertices as well.

It is important to remove as many vertices as possible in order to compute real-time edits on the mesh; on the other hand the coarse mesh should contain the structural details that need to be edited.

- In a second step the surface is smoothed using the discrete fairing operator. Either the membrane or the thin plate energy can be minimized. The difference vectors between the original position of the vertices and their position after the smoothing are stored in local frames.
- Next, the user needs to specify the area where the edit operation has to take place and how the edit will modify the mesh.
- Once the edit has been computed, the detail vectors storing high frequency information of the mesh are re-introduced.
- Finally, the mesh can be refined, re-introducing the vertices that have been removed in the first step of the algorithm.

8.3 Multiresolution Signal Processing for Meshes

In [46] Guskov, Sweldens and Schroeder present a new signal processing algorithm that can be used to fair meshes with arbitrary microtopology, which is based on the technical report [40] of Guskov. The most important difference between this new approach and the approach of Taubin [86] and the approach of Kobbelt [48] is that the operators that approximate the Laplacian also consider the geometric information and not only the topological information.

In the first subsection the difference operator of first and second order will be constructed for meshes described in a functional setting, which means that each vertex position is described as

$$\boldsymbol{x_i} = (u_i, v_i, g(u_i, v_i)) \qquad \text{(EQ 8.42)}$$

where u_i and v_i correspond to the global parameter position of the vertex $\boldsymbol{x_i}$ and $g(u_i, v_i)$ is a function which describes the z-values.
The results produced in this setting will then be generalized to the standard 3D setting, where two-manifold meshes do not have a natural global parametrization.

Next, an irregular subdivision scheme based on the operators described above will be introduced, which allows to construct smooth surfaces with irregular connectivity.
An extension of the Burt-Adelson pyramid based on the subdivision and difference operators will also be derived. This scheme allows to compute smooth downsampling while storing the information needed to invert the process.

8.3.1 Difference Operators in the Functional Setting

In order to introduce the first and second order difference operators some notation needs to be defined. The components needed for the construction of these operators are given in Figure 8.11:

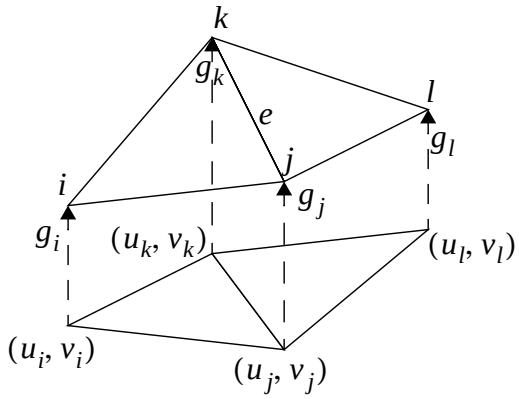


Figure 8.11 Triangles in the functional setting

The *first order divided difference* of the function $g(u, v)$ for a triangle $f = \{i, j, k\}$ is defined as the gradient of $g(u, v)$:

$$\nabla_f g(u, v) = \left(\frac{\partial g(u, v)}{\partial u}, \frac{\partial g(u, v)}{\partial v} \right) = (g_u(u, v), g_v(u, v)) \qquad \text{(EQ 8.43)}$$

The gradient of the function $g(u, v)$, function only defined at the vertex positions, at the triangle f can be computed solving the linear system of equations:

$$\begin{bmatrix} 1 & u_i & v_i \\ 1 & u_j & v_j \\ 1 & u_k & v_k \end{bmatrix} \cdot \begin{bmatrix} C \\ g_u(f) \\ g_v(f) \end{bmatrix} = \begin{bmatrix} g(u_i, v_i) \\ g(u_j, v_j) \\ g(u_k, v_k) \end{bmatrix} \qquad \text{(EQ 8.44)}$$

which results in

$$g_u^1(f) = \frac{(g(u_j, v_j) - g(u_i, v_i))(v_k - v_i) - (g(u_k, v_k) - g(u_i, v_i))(v_j - v_i)}{A_{[i, j, k]}} \qquad \text{(EQ 8.45)}$$

$$g_v^1(f) = \frac{(g(u_k, v_k) - g(u_i, v_i))(u_j - u_i) - (g(u_j, v_j) - g(u_i, v_i))(u_k - u_i)}{A_{[i, j, k]}} \qquad \text{(EQ 8.46)}$$

where

$$A_{[i,\,j,\,k]} = (u_j - u_i)\cdot(v_k - v_i) - (u_k - u_i)\cdot(v_j - v_i) \qquad \text{(EQ 8.47)}$$

It should be noted that the gradient in the interior of a triangle f of the mesh remains constant, since the function $g(u, v)$ is only defined for the three vertices that define the triangle and the function $g(u, v)$ is assumed to be piecewise linear in the interior of f.

Next the author defines the *second order differences*. This can be computed as the difference between the first order divided difference associated with two neighbor triangles. Equivalently it can be defined as the difference of the normals of two neighbor triangles, where the normal of a triangle $f = \{i, j, k\}$ is defined as

$$\boldsymbol{n}_f = (-g_u(f), -g_v(f), 1) \qquad \text{(EQ 8.48)}$$

This results in a vector that describes the second order difference

$$g_u^2(e) = g_u^1(f_2) - g_u^1(f_1) \qquad \text{(EQ 8.49)}$$

$$g_v^2(e) = g_v^1(f_2) - g_v^2(f_1) \qquad \text{(EQ 8.50)}$$

where $f_1 = \{i, j, k\}$, $f_2 = \{l, k, j\}$ and e is the edge shared by the two triangles (see Figure 8.11). Eq. 8.49 and Eq. 8.50 can be expanded to:

$$\begin{aligned} g_u^2(e) = {} & \frac{(v_j - v_k)}{A_{[i,\,j,\,k]}}\cdot g(u_i, v_i) + \left(\frac{v_k - v_l}{A_{[l,\,j,\,k]}} - \frac{v_k - v_i}{A_{[i,\,j,\,k]}}\right)\cdot g(u_j, v_j) + \\ & + \frac{(v_j - v_k)}{A_{[l,\,j,\,k]}}\cdot g(u_l, v_l) + \left(\frac{v_l - v_j}{A_{[l,\,j,\,k]}} - \frac{v_i - v_j}{A_{[i,\,j,\,k]}}\right)\cdot g(u_k, v_k) \end{aligned} \qquad \text{(EQ 8.51)}$$

$$\begin{aligned} g_v^2(e) = {} & \frac{(u_k - u_j)}{A_{[i,\,j,\,k]}}\cdot g(u_i, v_i) + \left(\frac{u_l - u_k}{A_{[l,\,j,\,k]}} - \frac{u_i - u_k}{A_{[i,\,j,\,k]}}\right)\cdot g(u_j, v_j) + \\ & + \frac{(u_k - u_j)}{A_{[l,\,j,\,k]}}\cdot g(u_l, v_l) + \left(\frac{u_j - u_l}{A_{[l,\,j,\,k]}} - \frac{u_j - u_i}{A_{[i,\,j,\,k]}}\right)\cdot g(u_k, v_k) \end{aligned} \qquad \text{(EQ 8.52)}$$

Finally, the curvature at an edge e can be estimated by computing the signed norm of the vector components defined in Eq. 8.51 and Eq. 8.52:

$$D_e^2 g = c_{e,\,i}\cdot g(u_i, v_i) + c_{e,\,j}\cdot g(u_j, v_j) + c_{e,\,k}\cdot g(u_k, v_k) + c_{e,\,l}\cdot g(u_l, v_l) \qquad \text{(EQ 8.53)}$$

where

$$c_{e,\,i} = \frac{L_e}{A_{[i,\,j,\,k]}},\ c_{e,\,l} = \frac{L_e}{A_{[l,\,j,\,k]}} \qquad \text{(EQ 8.54)}$$

$$c_{e,j} = \frac{L_e \cdot A_{[k,i,l]}}{A_{[i,j,k]} \cdot A_{[l,j,k]}}, c_{e,k} = \frac{L_e \cdot A_{[j,l,i]}}{A_{[i,j,k]} \cdot A_{[l,j,k]}} \quad \text{(EQ 8.55)}$$

and

$$L_e = \|(u_k - u_j, v_j - v_k, 0)\| \quad \text{(EQ 8.56)}$$

represents the length of the edge e projected onto the parameter space.

The operator computed in Eq. 8.53 can now be used in a relaxation algorithm to compute the new values of $g(u_i, v_i)$ for all vertices i, so that the energy

$$E = \sum_{\forall \text{ edge } e} (D_e^2 g)^2 \quad \text{(EQ 8.57)}$$

computed from Eq. 8.53 is minimized.

The relaxed value of a point $(u_i, v_i, g(u_i, v_i))$ is then computed as

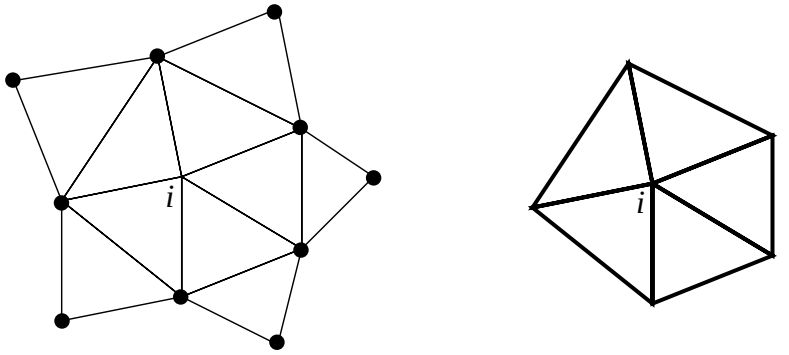
$$Rg_i = \min\left(\sum_{e \in \varepsilon_2(i)} (D_e^2 g)^2\right) = \sum_{j \in V_2(i)} w_{i,j} \cdot g_j \quad \text{(EQ 8.58)}$$

where

$$w_{i,j} = -\frac{\sum_{\{e \in \varepsilon_2(i) | j \in \omega(e)\}} c_{e,i} \cdot c_{e,j}}{\sum_{e \in \varepsilon_2(i)} (c_{e,i})^2} \quad \text{(EQ 8.59)}$$

The set $V_2(i)$ is defined as all the vertices on the 1-ring with flaps of the vertex i, as shown on the left of Figure 8.12, the set $\omega(e)$ represents the indices $\{i, j, k, l\}$ in

Eq. 8.53, and $\varepsilon_2(i)$ is the set of all the edges on the 1-ring of the vertex i, as shown on the right of Figure 8.12.



$V_2(i) = \{\bullet\}$ $\qquad$ $\varepsilon_2(i) = \{—\}$

a) $\qquad$ b)

Figure 8.12 Parameters of Eq. 8.58
a) Vertices on the one-ring with flaps of *i*
b) Edges on the one-ring of *i*

8.3.2 Extension to the Non-Parametrized Meshes

The operators derived in Section 8.3.1 are not directly applicable on arbitrary meshes, since in general no global parametrization of meshes is known. Furthermore if such a parametrization would exist, the three components of the vertices, x, y, and z would have to be faired separately.

It is however possible to generalize Eq. 8.58 slightly and to construct a new operator that smooths non-parametrized mesh vertices $\boldsymbol{x_i}$:

$$\boldsymbol{R x_i} = \sum_{j \in V_2(i)} w_{i,j} \cdot \boldsymbol{x_j} \qquad \text{(EQ 8.60)}$$

In order to correctly compute the weights $w_{i,j}$ the value $c_{e,i}$ (see Eq. 8.59) needs to be computed. This operation requires a local parametrization of the mesh. Fortunately the support of this parametrization is very small: it just has to cover the two triangles that share the edge e, since the operator Eq. 8.53 has exactly this support. A simple parametrization that does not introduce distortion is presented in Figure 8.13: the edge between the

two triangles is used as a hinge, and one triangle is rotated on it until the two triangles lie on the same plane. This parametrization is called *hinge map*.

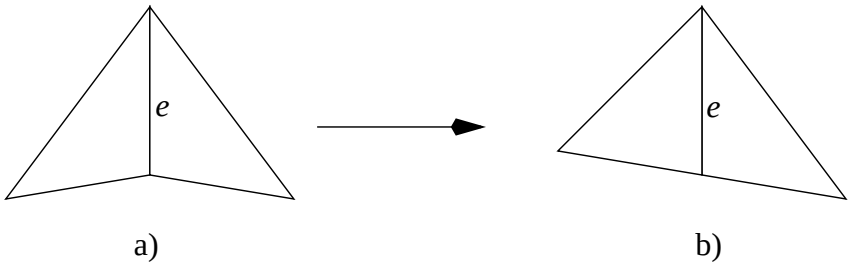


Figure 8.13 Local parametrization using hinges maps
a) Two triangles in 3-space
b) The hinge map rotate one triangle in the plane defined by the other triangle

This simple parametrization allows one to compute the values described in Eq. 8.54 and Eq. 8.55, basically four areas and the length of the edge e.

Once the weights $w_{i,j}$ have been computed, the same relaxation step can be performed on the mesh vertices $\boldsymbol{x_i}$, until the mesh has been faired.

8.3.3 Non-Uniform Subdivision

The second order difference operator can also be used to construct a so-called *non-uniform subdivision scheme*. This non-uniform scheme is defined over a mesh based decimation scheme; in the original paper the authors used a simplified progressive mesh algorithm based on half-edge collapse, where one vertex is removed from the mesh by collapsing it into another neighbor vertex.

The flow of the algorithm is outlined below:

- the first step consists in constructing a progressive mesh representation from an input mesh. An important difference between standard subdivision schemes and this new scheme is that in the former case the input mesh is coarse and it is refined, whereas in the latter case the input mesh is a fine mesh that is coarsified through a decimation algorithm.
  The result of this step is a sequence of meshes

$$\boldsymbol{Q^m} \rightarrow \boldsymbol{Q^{m-1}} \rightarrow \ldots \rightarrow \boldsymbol{Q^{n_0}} \qquad \text{(EQ 8.61)}$$

- The subdivision algorithm starts with the coarse mesh $\boldsymbol{Q^{n_0}}$. Each subdivision step introduces a new vertex, defined by the vertex split operation. Going from a subdivision level $\boldsymbol{Q^{n-1}}$ to the new level $\boldsymbol{Q^n}$ is accomplished in the following steps:
  - The position of the vertex being introduced is computed using Eq. 8.60:

$$\boldsymbol{x}_{\boldsymbol{n}}^{(n)} = \sum_{j \in V_2(n)} w_{n,j}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{j}}^{(n-1)} \quad \text{(EQ 8.62)}$$

- The position of the vertices on the one-ring of the newly introduced vertex n are also updated using Eq. 8.60:

$$\boldsymbol{x}_{\boldsymbol{j}}^{(n)} = \sum_{k \in V_2(j)\setminus\{n\}} w_{j,k}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{k}}^{(n-1)} + w_{j,n}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{n}}^{(n)} \quad \text{(EQ 8.63)}$$

- All the other vertices in the mesh do not move:

$$\boldsymbol{x}_{\boldsymbol{j}}^{(n)} = \boldsymbol{x}_{\boldsymbol{j}}^{(n-1)} \quad \text{(EQ 8.64)}$$

8.3.4 Burt-Adelson Pyramid

The Burt-Adelson pyramid can be used in conjunction with the second order difference operator to construct a hierarchical representation of a mesh. The removal of a vertex n from an input mesh $\boldsymbol{Q^n}$ is computed in four steps:

- *Pre-smoothing*: downsampling a signal without first pre-smoothing it usually introduces aliasing effects. It is therefore necessary to smooth the one-ring neighborhood of the vertex n:

$$\boldsymbol{x}_{\boldsymbol{j}}^{(n-1)} = \sum_{k \in V_2(n)\setminus\{n\}} w_{j,k}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{k}}^{(n)} \quad \text{(EQ 8.65)}$$

- *Downsampling*: the vertex n is removed from the mesh. The removal could be performed with the half-edge collapse operation.
- *Subdivision*: the subdivision scheme presented in Section 8.3.3 can be used as a refinement operator. The mesh is refined by introducing the vertex n back in the mesh and computing its position using Eq. 8.60 as well as the new position of the vertices on the one-neighborhood of n.
  This information will be used to compute the detail information in the next step.

  Using the subdivision rule Eq. 8.62 the position of the vertex n can be estimated as:

$$\boldsymbol{s}_{\boldsymbol{n}}^{(n)} = \sum_{k \in V_2(n)} w_{n,k}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{k}}^{(n-1)} \quad \text{(EQ 8.66)}$$

  and using the subdivision rule Eq. 8.63 the position of the neighbor vertices of n can be estimated as:

$$\boldsymbol{s}_{\boldsymbol{j}}^{(n)} = \sum_{k \in V_2(j)\setminus\{n\}} w_{j,k}^{(n)} \cdot \boldsymbol{x}_{\boldsymbol{k}}^{(n-1)} + w_{j,n}^{(n)} \cdot \boldsymbol{s}_{\boldsymbol{n}}^{(n)} \quad \text{(EQ 8.67)}$$

- *Computation of the details*: the detail values needed to invert the downsampling step, one for the vertex n and one for every neighbor of n, are stored in a local frame $F_j^{(n-1)}(\boldsymbol{x})$:

$$\boldsymbol{d}_j^{(n)} = F_j^{(n-1)}(\boldsymbol{x}_j^{(n)} - \boldsymbol{s}_j^{(n)}) \quad \text{(EQ 8.68)}$$

These four steps are presented graphically in Figure 8.14:

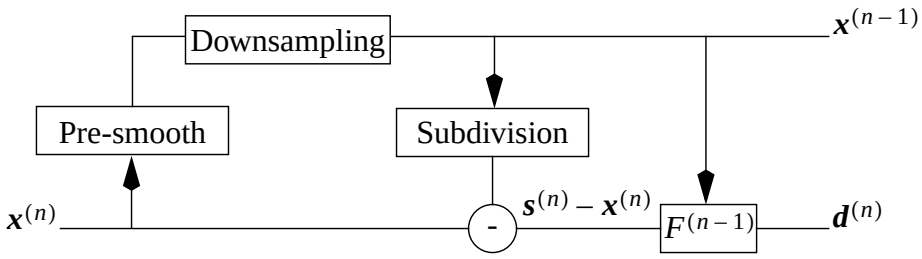


Figure 8.14 Extended Burt-Adelson pyramid scheme

The parameters $\boldsymbol{x}_i^{(j)}$ and in $\boldsymbol{d}_i^{(j)}$ store enough information to reconstruct the original mesh. The reconstruction process is described by the following steps:

- The vertex n is reintroduced, and its geometric position is guessed using the non-uniform subdivision scheme

$$\boldsymbol{s}_n^{(n)} = \sum_{j \in V_2(n)} w_{n,j}^{(n)} \cdot \boldsymbol{x}_j^{(n-1)} \quad \text{(EQ 8.69)}$$

  The position of the vertices on the one-neighborhood of n are also smoothed using the rules constructed in Section 8.3.3:

$$\boldsymbol{s}_j^{(n)} = \sum_{k \in V_2(j) \setminus \{n\}} w_{j,k}^{(n)} \cdot \boldsymbol{q}_k^{(n-1)} + w_{j,n}^{(n)} \cdot \boldsymbol{s}_n^{(n)} \quad \text{(EQ 8.70)}$$

- In a second step the detail values stored in $\boldsymbol{d}_j^{(n)}$ are added back to the vertex n and the vertices on its one-neighborhood:

$$\boldsymbol{q}_j^{(n)} = \boldsymbol{s}_j^{(n)} + (F_j^{(n-1)})^{-1}(\boldsymbol{d}_j^{(n)}) \quad \text{(EQ 8.71)}$$

8.4 Implicit Fairing Using Curvature Flow

M. Desbrun et al. presented in [24] some interesting enhancements to the basic smoothing strategy constructed by Taubin presented in Section 8.1. These enhancements allow to construct a more robust algorithm to smooth meshes with arbitrary topology and reduce the need of human supervision during the smoothing process. Furthermore the operators

presented in this paper allow to obtain better results, both with respect to the quality of the smoothing and the shape of the triangles in the mesh.

In the next few subsections the following improvements will be discussed:

- Use of an implicit solver in the smoothing process
- A simple strategy to guarantee the preservation of the volume of a mesh during the smoothing process
- A new geometry-based operator based on the Laplacian
- A new operator based on the concept of curvature flow

8.4.1 Implicit Fairing

The fairing process described by Taubin using Eq. 8.9 corresponds to integrating the diffusion equation using an explicit Euler scheme. The equation for the smoothing step is shown in Eq. 8.72

$$\boldsymbol{x}^{(k+1)} = (I - \lambda dtK)\boldsymbol{x}^{(k)} \qquad \text{(EQ 8.72)}$$

The inherent problem with this approach is that explicit methods behave poorly if the system is stiff, and in order to converge to the correct solution it is necessary to use very small time steps. The choice of the time steps is dependent on the input mesh, more precisely it is dependent on the length of the edges of the input mesh, and therefore manual intervention is necessary to compute good-quality results. Since the time steps are required to be small usually many iterations are needed to converge to the smooth mesh.

The problems of the previous approach can be solved by using implicit integration. It is well known that implicit integration methods are more stable than explicit methods. As a consequence fewer iterations will be needed to converge to the solution, since larger time steps dt can be used. In an implicit method the approximation of the Laplacian $\Delta\boldsymbol{x}$ is computed using the geometric information at iteration $(k + 1)$ and not the information at iteration k. If this concept is applied to Eq. 8.72 the following system of equations can be derived:

$$(I + \lambda dtK)\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} \qquad \text{(EQ 8.73)}$$

From an algorithmic point of view, the most important difference between Eq. 8.72 and Eq. 8.73 is that while Eq. 8.72 can be evaluated very easily using a matrix-vector multiplication, a system of equations must be solved to compute the result of Eq. 8.73. Although the implicit method can require more time to compute an iteration, the authors found out that the algorithm is usually faster, since fewer iterations are required to converge to the solution.

In Figure 8.15 a simple comparison of the two integration methods is presented. For the same number of smoothing iterations the implicit method generates better results, since it is more stable and can work with larger time steps.

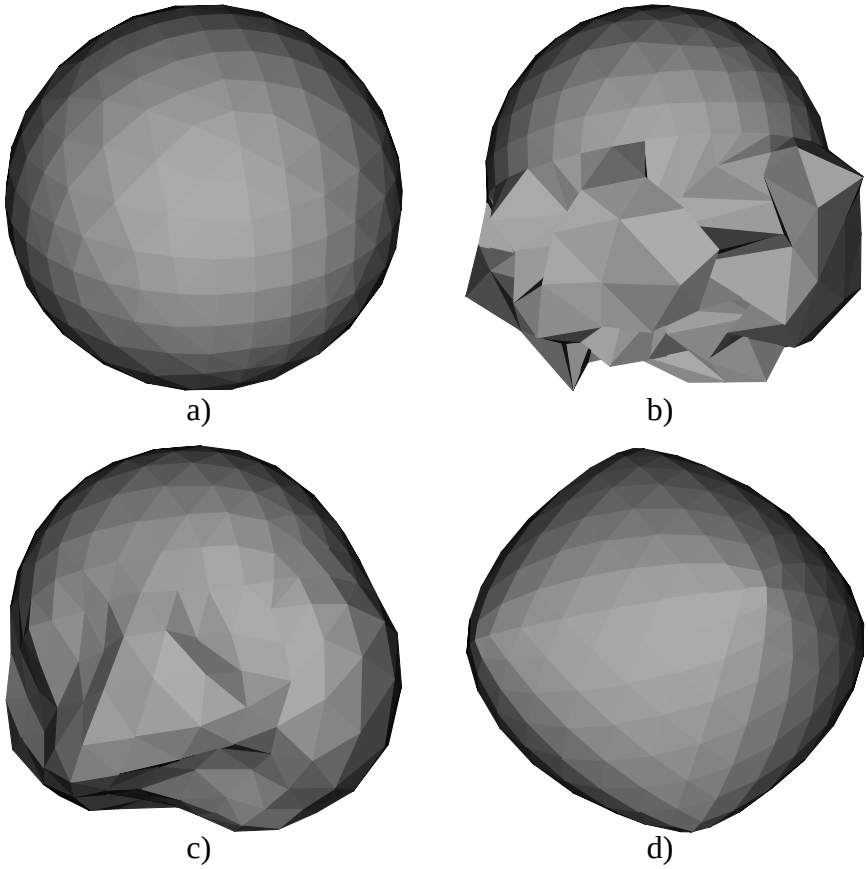


Figure 8.15 Comparison of explicit and implicit integration
a) Original triangular mesh
b) Noise is added to the mesh a)
c) Result after five smoothing steps using an explicit method
d) Result after five smoothing steps using an implicit method

8.4.2 Volume Preservation

In his original paper Taubin avoids shrinking in the mesh by applying an un-shrinking step, as shown in Eq. 8.10. In general this approach gives good results, and no shrinking effects occur during smoothing. The major drawback of this scheme is that the volume is not preserved in any way: the volume can vary from smoothing step to smoothing step.

Desbrun proposes in [24] a simple strategy that allows to maintain the original volume of the mesh using a simple linear time algorithm. The author computes the volume of all oriented pyramids defined by the triangles in the mesh and by a point in space (the origin for example). This volume can be computed easily using Eq. 8.74

$$V = \frac{1}{6} \cdot \sum_{t \in \text{ Triangles}} \boldsymbol{g_t} \cdot \boldsymbol{N_t} \qquad \text{(EQ 8.74)}$$

where

$$\boldsymbol{g_t} = \frac{(\boldsymbol{x_t^1} + \boldsymbol{x_t^2} + \boldsymbol{x_t^3})}{3} \quad \text{(EQ 8.75)}$$

and

$$\boldsymbol{N_k} = (\boldsymbol{x_t^2} - \boldsymbol{x_t^1}) \wedge (\boldsymbol{x_t^3} - \boldsymbol{x_t^1}) \quad \text{(EQ 8.76)}$$

$\boldsymbol{x_t^1}, \boldsymbol{x_t^2}, \boldsymbol{x_t^3}$ represent the three vertices that define the triangle t.

The volume can then be preserved during the smoothing process by first computing the initial volume of the mesh V_0, and by scaling the vertex positions of the mesh during iteration k by $\beta = \left(\frac{V_0}{V_k}\right)^{1/3}$.

8.4.3 A Geometry-Based Laplacian Operator

Desbrun et al. identified a fundamental problem in the formulation of the Laplacian operator defined by Taubin in [86] and by Kobbelt in [48]: their definition of the Laplacian is based only on topological information and not on geometric information. This problem has been identified by Guskov et al. as well who also presented a new operator (see Section 8.3).
The most important drawback that purely topological methods have is that if the mesh has non-uniform density, then after smoothing the mesh will lose some features, such as symmetry (see the original paper for pictures describing the problem in more detail).

The authors extended the basic Laplacian operator by considering edge lengths, and not only the valence of the vertices. The resulting formula is shown in Eq. 8.77

$$\Delta \boldsymbol{x_i} = \frac{2}{E} \cdot \sum_{j \in N_1(\boldsymbol{x_i})} \frac{\boldsymbol{x_j} - \boldsymbol{x_i}}{|e_{i,j}|} \quad \text{(EQ 8.77)}$$

where

$$E = \sum_{j \in N_1(\boldsymbol{x_i})} |e_{i,j}| \quad \text{(EQ 8.78)}$$

$N_1(\boldsymbol{x_i})$ represents the vertices of the star of $\boldsymbol{x_i}$ and $|e_{i,j}|$ represents the length of the edge $e_{i,j}$ defined between the vertices $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$.

Note that if all the edge lengths $|e_{i,j}|$ are equal to 1 then Eq. 8.77 reduces to the operator constructed by Taubin.

8.4.4 The Curvature Flow Operator

Perhaps the most interesting extension presented in the paper is the new operator based on the concept of curvature flow. The operator that was presented in Section 8.4.3 still has one shortcoming: if the algorithm is applied to a flat triangular mesh, where all triangles lie in a plane, the smoothing step will move the vertices in the plane. The mesh generated is still flat, but the shape of the triangles has changed during the smoothing step.
The curvature flow operator is formulated in such a way as to avoid these changes in the shape of the triangles. This is accomplished by moving the vertices only along the surface normal $\boldsymbol{n}$ with speed proportional to the mean curvature $\bar{\kappa}$, as shown in

$$\frac{\partial \boldsymbol{x_i}}{\partial t} = -\bar{\kappa}_i \cdot \boldsymbol{n_i} \qquad \text{(EQ 8.79)}$$

The mean curvature $\bar{\kappa}_i$ is defined as

$$\bar{\kappa}_i = \frac{(\kappa_1 + \kappa_2)}{2} \qquad \text{(EQ 8.80)}$$

The authors used the following definition of curvature normal to build the operator for meshes with arbitrary connectivity:

$$\frac{\nabla A}{2A} = \bar{\kappa} \cdot \boldsymbol{n_i} \qquad \text{(EQ 8.81)}$$

where A is the area of a small region surrounding $\boldsymbol{x_i}$, and ∇A is the derivative of A with respect to the coordinates of $\boldsymbol{x_i}$. If the area A is assumed to be the union of the triangles that surrounds $\boldsymbol{x_i}$, then Eq. 8.81 can be expanded into

$$-\bar{\kappa} \cdot \boldsymbol{n_i} = \frac{1}{4A} \cdot \sum_{j \in N_1(i)} (\cot(\alpha_j) + \cot(\beta_j)) \cdot (\boldsymbol{x_j} - \boldsymbol{x_i}) \qquad \text{(EQ 8.82)}$$

where α_j and β_j represents the angles opposite to the edge $e_{i,j}$, as shown in Figure 8.16.

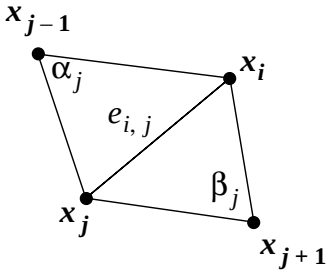


Figure 8.16 Notation used in Eq. 8.82

8.5 Evaluation

In this section the fairing operators presented in this section will be evaluated using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: The fairing operators presented in this section only require a local parametrization on a very small neighborhood, so they can be applied to any two-manifold mesh, and no subdivision connectivity is required. If the two-manifold has boundaries, special operators need to be constructed to avoid shrinkage.
- *Computation of surface-surface intersections*: The fairing operators described in this section do not encode the topology of the surface they are applied on, instead they are normally used to encode the geometric information. The goal of these operators is to fair meshes and generate smooth approximations of non-smooth meshes. The representations constructed with these operators do not contain the information needed to compute intersection curves, and it is therefore necessary to use an external data structure such as a hierarchical bounding box to accelerate the computations.

  Kobbelt's and Guskov's representations are based on progressive meshes, and therefore they have the same problem the standard progressive meshes have: too many triangles have to be tested to find the intersection. For more information refer to Section 7.5.
- *Scalable representation*: *Taubin*'s and *Desbrun*'s representations are not scalable, since they do not construct a multiresolution representation of the surface. The fairing operators affect only the geometry of a surface, but not the connectivity.
  - If the surface is stored in a compact data structure then this representation is also compact, since it only modifies the geometry of the surface.
  - The representation can only return a faired mesh, but it does not compute a coarse approximation of the mesh.
  - All the fairing algorithms have a linear complexity $O(k \cdot n)$, where k is the number of iterations used in the fairing step.

  Kobbelt's representations use a fairing operator in conjunction with a mesh simplification algorithm such as progressive mesh. As a result the author is able to construct scalable multiresolution representations of surfaces that satisfy these properties:
  - The representation is compact, since it is based on progressive mesh. Faired vertices as well as the detail information stored by the progressive mesh representation are stored in local frames to allow multiresolution editing of surfaces, and this representation does not introduce an overhead.
  - The representation is capable of building faired approximations of the mesh. In order to do that Kobbelt first extracts an approximation from the progressive mesh representation, and then he fairs it using the umbrella algorithm.
  - The construction of the progressive mesh representation takes at least $O(n \cdot \log n)$, and the reconstruction can be performed in linear time. The umbrella algorithm has a linear complexity of $O(k \cdot n)$, where k represents the number of smoothing iterations on the mesh.

However this approach is not well suited for an adaptive representation, since the fairing process is not integrated into the progressive mesh representation, it is a post-process applied to the mesh extracted from the progressive mesh.

Guskov's representation also used a fairing operator in conjunction with a progressive mesh representation, but he constructs a non-uniform subdivision that integrates the smoothing in the progressive mesh.

- The representation Guskov describes in [46] is not compact: the topology is stored compactly using the PM representation, but the geometry is not. Analyzing the encoding step of the geometry, described by Eq. 8.65 through Eq. 8.68, shows immediately that in order to store the geometry of a single vertex being removed, $1 + V_1(j)$ details values must be stored, where $V_1(j)$ represent the valence of the vertex j. In [18], Daubechies and Guskov describe how it would be possible to construct a compact critically sampled representation, but no practical solution to the problem is provided.
- The representation described in Section 8.3 is capable to construct approximations of full resolution surfaces. This is accomplished constructing an operator that inverts the representation that is constructed using the Burt-Adelson pyramid described by Figure 8.14.
- The construction of the representation has a minimum complexity of $O(n \cdot \log n)$, which is the time required to construct a PM representation. The encoding of the geometry can be performed very efficiently and it does not increase the complexity of the construction algorithm. An approximation can be extracted from the representation in $O(n)$ time.

- *Modeling of non-manifold singularities, tears and cracks*: The Laplacian is not defined at non-manifold singularities, and therefore the representations presented in this section are not able to model them.

  Taubin and *Kobbelt* assume the vertices on the star of a vertex can be parametrized regularly using Eq. 8.36. This parametrization assumes that the operator is applied over a manifold surface.

  Guskov's representation does not require a regular parametrization, since the idea is to use not only topological information, but also geometric information. Since the only parametrization the author requires is very local and can be constructed in many ways, such as using the hinge map, at first it seems that this representation might be powerful enough to represent these singularities. This is not true however, since in the construction of the weights used in Eq. 8.60 the author assumes that an edge bounds at most only two triangles.

  Desbrun also assumes implicitly that his operators are applied on two-manifold meshes, since he makes use of primitives such as the Laplacian and normal which are not well defined at non-manifold singularities.

  None of the three representations can represent curves in the surface with their data structure. Since the surfaces are represented as piecewise linear meshes, it is possible to

construct a separate data structure that stores the embedding of the piecewise linear curve in the mesh.

- *Error modeling*: None of the representations described in this section construct an error model, since the goal is not to construct the most accurate approximations of a surface, but to smooth piecewise linear meshes and eventually to construct a multiresolution representation of meshes based on a smoothing operator.

  An interesting error that could be analyzed further describes how well the different operators described in Eq. 8.6, Eq. 8.37, and Eq. 8.53 approximate the Laplacian. Since the Laplacian is not well defined in this setting the evaluation of the quality of the operators remains an outstanding problem.

- *Smoothness of the surface*: The four representations are by definition not smooth, since they operate on a piecewise linear approximation of smooth surfaces. However the goal of the operators is to minimize the discrete Laplacian by modifying the geometry of the meshes, which means they are constructing the smooth approximation of a mesh using the same number of vertices and the same connectivity. This corresponds to projecting a piecewise linear mesh to the space of the meshes with lower frequency.

- *Multiresolution editing*: *Taubin*'s and *Desbrun*'s representations do not construct a multiresolution representation of surfaces, and they are therefore unable to edit the surface at different levels of resolution.

  Kobbelt constructs a representation based on a smoothing operator similar to the operator built by Taubin, but it uses it in a multiresolution representation based on progressive mesh. The resulting representation allows to compute edit operations at any level of resolution well. The quality of the edits is further enhanced by storing the geometric information in local frames.

  The representation based on the Burt-Adelson pyramid constructed by *Guskov* also allows multiresolution edits, since it is based on the PM representation and encodes the geometric information in local frames.

- *Surface fitting*: Non of the existing surface fitting algorithms construct any of the representation described in this section automatically. Any fitting scheme that generates a single resolution mesh can be used in *Taubin*'s and *Desbrun*'s representations to construct the input mesh that will then be smoothed with their operators. The fitting strategy developed by Hoppe in his thesis can be used to construct the basic PM representation that will then be used by Kobbelt and Guskov in their representations.

- *Support of local high variation in the curvature of the surface*: All four representations work on piecewise linear representation of meshes, and they are therefore capable to model high curvature in the surface. The smoothing operators are then used to remove these regions of high curvature. If curvature must be maintained in some specific region of the mesh, then it is necessary not to apply the smoothers there.

- *Changes of the surface over time*: This family of representations can model changes in the surface over time only if the changes do not affect the topological information too much:

  1. If the connectivity of the mesh does not change over time, then the geometric changes over time can be modeled easily by defining a function $f(t)$ at each ver-

tex instead of a single value (x, y, z). At a time t_i the value of the vertex would be extracted from the function: $f(t_i) = (x_i, y_i, z_i)$.

2. If the connectivity changes over time, but the macro-topology remains the same, then it would be possible to compute a remeshing of the surface at all times t_i to generate surfaces with the same connectivity. Once this has been done the solution presented in 1. can be applied.
3. If the micro-topology of the mesh changes as well, then the only viable solution is to keep different representations at each point in time t_i.

The operators presented in this section could also be used to compute antialiasing in time, thus generating smooth changes of the surfaces over time.

9 Representations Commonly Used in Geoscience

In this last section two of the most used representations in geoscience will be described:

- *Discrete Smooth Interpolation*: this algorithm allows to smoothly fit data through a discrete set of vertices in a surface. The main advantage of this scheme is the ability to apply a wide number of constraints during the fitting step. This allows to enforce geological constraints and therefore to generate meaningful representations.
- *Quadtrees*: this classic representation is used in different fields to represent data that can be stored in regular grids. It is widely used for terrain visualization and it has been used in geologic applications. Quadtree representations can often be defined as a wavelet representation using either the *Haar* or *lazy wavelet* basis functions.

9.1 Discrete Smooth Interpolation

The goal of the Discrete Smooth Interpolation (DSI) algorithm, presented by J. Mallet in [59] is to construct an approximation of the value of a function $f(i)$ for all the vertices i in a mesh where $f(i)$ is not defined. The algorithm presented in the paper works on meshes built from polygonal faces. We will limit our description to triangular meshes.

The problem can be formulated as follow: the DSI algorithm will minimize the criterion specified in Eq. 9.1

$$E(f) = R(f) + \rho(f) \qquad \text{(EQ 9.1)}$$

The first term in Eq. 9.1, $R(f)$, is a roughness criterion. This criterion is used to select one of the infinite number of surfaces that satisfy the second criterion $\rho(f)$. The approximation of the function $f(i)$ at all vertices i where $f(i)$ is unknown will then minimize this criterion. The roughness is defined as

$$R(f) = \sum_{i \in V} \mu(i) \cdot R(f|i) \qquad \text{(EQ 9.2)}$$

The term $\mu(i)$ represents a non-negative weight associated with each vertex i, and $R(f|i)$, the local roughness criterion at the vertex i, is defined as

$$R(f|i) = \left| \sum_{j \in N(i)} \nu(i, j) \cdot f(j) \right|^2 \qquad \text{(EQ 9.3)}$$

The set $N(i)$ represents the set of vertices on the k-th star of the vertex i, where k is a user-specified parameter. A vertex j is in the k-th star of i if there is a path $i, p_0, \ldots p_m, j$ between vertices i and j of length smaller or equal $k+1$. The term $\nu(i, j)$ represents a weighing coefficient associated with the pair of vertices i and j.

The second term in Eq. 9.1, $\rho(f)$, describes a set of linear constraints that must be satisfied during the interpolation step. Linear constraints can be specified using a linear system of equations such as

$$A \cdot \begin{bmatrix} f(0) \\ \dots \\ f(n-1) \end{bmatrix} \cong \boldsymbol{b} \qquad \text{(EQ 9.4)}$$

Since there might be more constraints than degrees of freedom, it is not always feasible to satisfy all the constraints, and therefore the symbol $\cong$ was used in Eq. 9.4. Furthermore it is possible to assign a weight ω_i, a positive scalar value, to each constraint that describes its importance; as a result it is desirable to minimize

$$\omega_i \cdot \left| \text{row}\,(A, i) \cdot \begin{bmatrix} f(0) \\ \dots \\ f(n-1) \end{bmatrix} - b_i \right|^2 \qquad \text{(EQ 9.5)}$$

for each row i of the matrix A and vector $\boldsymbol{b}$.

The violation of the set of constraints described in Eq. 9.4, i.e. how much Eq. 9.5 differs from zero for each constraint, is expressed by $\rho(f)$, as illustrated by

$$\rho(f) = \sum_{i=0}^{C-1} \omega_i \cdot \left| \text{row}\,(A, i) \cdot \begin{bmatrix} f(0) \\ \dots \\ f(n-1) \end{bmatrix} - b_i \right|^2 \qquad \text{(EQ 9.6)}$$

The term C describes the number of constraints applied to the vertices $0, \dots, n-1$.

Now that Eq. 9.1 has been fully described it is necessary to solve for the function value $f(i)$ for all vertices i where the function is not defined. In order to accomplish this it is helpful to reformulate Eq. 9.1 as

$$E(f) = \begin{bmatrix} f(0) & \dots & f(n-1) \end{bmatrix} \cdot Q \cdot \begin{bmatrix} f(0) \\ \dots \\ f(n-1) \end{bmatrix} + 2 \cdot \begin{bmatrix} f(0) & \dots & f(n-1) \end{bmatrix} \cdot L + c \qquad \text{(EQ 9.7)}$$

The matrix Q is defined as

$$Q = Q^1 + Q^2 \qquad \text{(EQ 9.8)}$$

The matrix Q^1 is derived from the expansion of Eq. 9.6

$$Q^1 = \sum_{i=0}^{C-1} \omega_i \cdot \mathrm{row}(A, i)^T \cdot \mathrm{row}(A, i) \qquad \text{(EQ 9.9)}$$

and the second matrix, Q^2 is derived from Eq. 9.2 and Eq. 9.3. Eq. 9.10 describes the value of the entry (i, j) in Q^2

$$Q^2_{i,j} = \sum_{k \in \{N(i) \cap N(j)\}} \mu(k) \cdot \nu(i, k) \cdot \nu(j, k) \qquad \text{(EQ 9.10)}$$

The matrix L is derived from Eq. 9.6 and it is defined as

$$L = \sum_{i=0}^{C-1} \omega_i \cdot \mathrm{row}(A, i)^T \cdot b_i \qquad \text{(EQ 9.11)}$$

The value of the constant c is also derived from Eq. 9.6 and it is defined as

$$c = \sum_{i=0}^{C-1} \omega_i \cdot b_i^2 \qquad \text{(EQ 9.12)}$$

The index associated with each vertex can be changed without affecting the mesh or the DSI algorithm. For the sake of simplicity it is assumed here that the value of $f(u)$ for the vertices $u \in \{0, \ldots, i-1\}$ is unknown, whereas the value of $f(k)$ for the vertices $k \in \{i, \ldots, n-1\}$ is known. Using this assumption it is possible to decompose the vectors and matrices defined in Eq. 9.7 as

$$Q = \begin{bmatrix} Q^{uu} & Q^{uk} \\ Q^{ku} & Q^{kk} \end{bmatrix} \qquad \text{(EQ 9.13)}$$

$$L = \begin{bmatrix} L^u \\ L^k \end{bmatrix} \qquad \text{(EQ 9.14)}$$

$$\begin{bmatrix} f(1) \\ \ldots \\ f(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{u}) \\ \boldsymbol{f}(\boldsymbol{k}) \end{bmatrix}, u \in \{0, \ldots, i-1\}, k \in \{i, \ldots, n-1\} \qquad \text{(EQ 9.15)}$$

The value of $f(u), u \in \{0, \ldots, i-1\}$ is then computed by solving

$$\frac{\partial E(f)}{\partial \boldsymbol{f}(\boldsymbol{u})} = 2 \cdot Q^{uu} \cdot \boldsymbol{f}(\boldsymbol{u}) + 2 \cdot Q^{uk} \cdot \boldsymbol{f}(\boldsymbol{k}) + 2 \cdot L^u \qquad \text{(EQ 9.16)}$$

which results in the linear system of equations described in Eq. 9.17

$$Q^{uu} \cdot \boldsymbol{f}(\boldsymbol{u}) = -Q^{uk} \cdot \boldsymbol{f}(\boldsymbol{k}) + L_u \qquad \text{(EQ 9.17)}$$

If a unique solution to the DSI problem is needed, then some constraints must be applied to the user-specified weights. The constraints have been formulated in

Theorem 5: *If the mesh is consistent, i.e. there is a vertex i in every connected component for which* $f(i)$ *is known, and if the global roughness criterion* $R(f)$ *satisfies*

- $\mu(i) > 0,\ i \in \{0, \ldots, n\}$
- $\nu(i, j) > 0,\ j \in N(i),\ j \neq i$
- $\nu(i, i) = -\sum_{j \in N(i) \setminus \{i\}} \nu(i, j) \neq 0$

then the DSI equation based on $R(f)$ *has a unique solution.*

For a proof of this theorem please refer to the original paper [59].

The evaluation of the Q and L matrices needed to evaluate the function $\boldsymbol{f}(\boldsymbol{u})$ at the set of vertices $u = \{0, \ldots, i-1\}$ is very expensive, with respect to both the computation and the storage requirements. In order to avoid this problem the author constructed an iterative method that converges to the solution. This is accomplished by observing that in order to satisfy

$$\frac{\partial E(f)}{\partial f(i)} = 0 \qquad \text{(EQ 9.18)}$$

$f(i)$ must satisfy the so-called *DSI-equation*

$$f(i) = -\frac{1}{M(i)} \left\{ \sum_{j \in N(i)} \left\{ \mu(j)\nu(i, j) \sum_{k \in N(j) \setminus \{i\}} \nu(j, k) f(k) \right\} + \sum_{j=0}^{C-1} \Gamma_j(i) \right\} \qquad \text{(EQ 9.19)}$$

where

$$M(i) = \sum_{j \in N(i)} \mu(j)(\nu(i, j))^2 + \sum_{j=0}^{C-1} \omega_i \cdot (A_{j,i})^2 \qquad \text{(EQ 9.20)}$$

$$\Gamma_j(i) = \omega_i \cdot A_{j,i} \cdot \left(\sum_{\substack{k=0 \\ k \neq i}}^{C-1} A_{j,k} \cdot f(k) - b_i \right) \qquad \text{(EQ 9.21)}$$

An iterative algorithm would then continuously update the value $f(i)$ at the vertex positions i where the value of $f(i)$ is not known.

If Theorem 5 is satisfied, then this iteration will converge to the unique solution of the system of equations Eq. 9.17 independently of the choice of the initial value of the $f(i)$s. A good choice of the initial values would however make the system converge faster.

A more detailed derivation of the iteration process is described in the original paper.

The theory presented in this section can also be extended to the case where more than one function must be interpolated. If the functions are independent, then the algorithm described in this section is adequate: every function is treated separately. If the functions are dependent however, like in the case where three functions are used to code the geometric information of the vertices (one function per x, y, and z coordinates), then the DSI algorithm must be extended. The extension is trivial and can be found in the original paper.

Some application of the DSI algorithm to geometric modeling include:

- Given a mesh, the exact position of some of the vertices, the approximate position of the remaining vertices and some vectorial constraints, the DSI algorithm will compute the geometric position of the vertices such that the vectorial constraints are satisfied as much as possible, and that the final position of the vertices is as close as possible to the approximate position specified by the user.
- In an interactive environment, a user can be presented with a geological surface. The user then interacts with the surface by changing the constraints and moving (editing) the position of some vertices. The DSI algorithm then computes the new shape of the mesh after the edits.

9.2 Restricted Quadtree Triangulation

Many approaches exists to construct a quadtree representation from a regular grid, most of them are optimized depending on the particular application. The quadtree representation is very compelling, since it is fairly simple, but powerful enough to represent large models.
In this survey we will discuss one representation constructed by R. Pajarola in [66]. This particular paper has been chosen, because it describes a specific implementation of the *Restricted Quadtree Triangulation* (RQT) that can handle very large datasets.

This section is organized as follow: in the first subsection the notion of a quadtree and of a restricted quadtree will be introduced. Next, a practical algorithm to extract an approximation from the quadtree will be presented, as well as a simple algorithm to construct an approximation of the error. An elegant and fast triangulation strategy will be discussed briefly.

9.2.1 Restricted Quadtree Triangulation (RQT)

As already mentioned, there are multiple ways to construct a quadtree, for example using the wavelet theory and the Haar or Lazy wavelet basis functions. The hierarchy used in [66] is different: a few levels of the quadtree representation are shown in Figure 9.1.

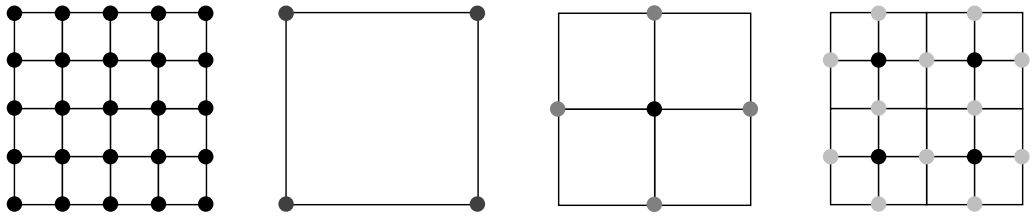


Figure 9.1 Quadtree hierarchy
a) Full-resolution grid
b) Level 0 of the quadtree (root node)
c) Level 1 of the quadtree
d) Level 2 of the quadtree (leaf nodes)

From now on, the term L_l described the set of vertices defined at level l in the quadtree, and the term L_l^c describes the vertices at level l that lie in the center of a quadtree node. The black dots in Figure 9.1 c) and d) are examples of such vertices

A triangulation of the quadtree hierarchy presented in Figure 9.1 is computed by first selecting some vertices from the quadtree, usually the most important ones according to an error criterion, and then the vertices are triangulated according to some rules.
For a quadtree triangulation to be restricted the levels of adjacent quadtree nodes must differ at most by one in the quadtree hierarchy. Using restricted triangulations is very useful, since it is possible to construct non-cracked triangulation using simple rules. In his paper Pajarola describes the restriction of the quadtree triangulation in terms of dependencies. Each vertex *i* in the grid depends on a set of vertices. These dependencies can be expressed in a dependency graph. If a vertex *i* is selected for the triangulation, then the triangulation is restricted if and only if the set of vertices associated with *i* in the dependency graph is also present in the triangulation. Figure 9.2 illustrates the dependency graph for the first two levels of the quadtree:

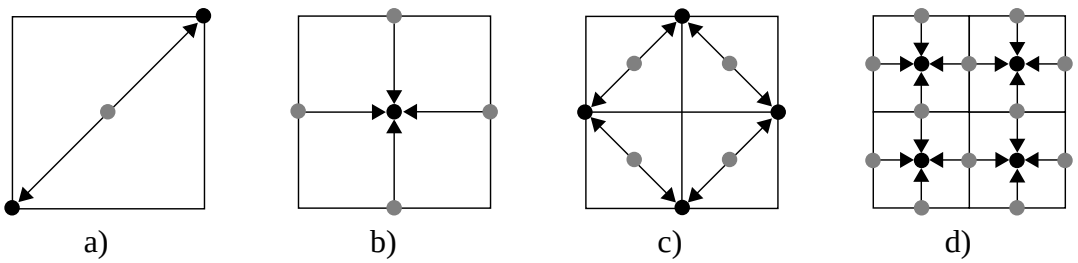


Figure 9.2 Dependency graph for the first two levels of the quadtree
a) Dependency of the center vertex of level *2*
b) Dependency of the remaining vertices of level *2*
c) Dependency of the center vertices of level *n*
d) Dependency of the remaining vertices of level *n*

If any of the vertices drawn in grey in Figure 9.2 is used in the triangulation, then the arrows indicate which vertices must be included as well in order to have a restricted quadtree triangulation. The dependency graph must then be followed recursively in order to guarantee that all the dependencies have been considered. Figure 9.3 shows a simple example where a set of vertices are selected for the triangulation in a), the dependency graph is applied to these vertices in b), and finally the new set of vertices is triangulated in c).

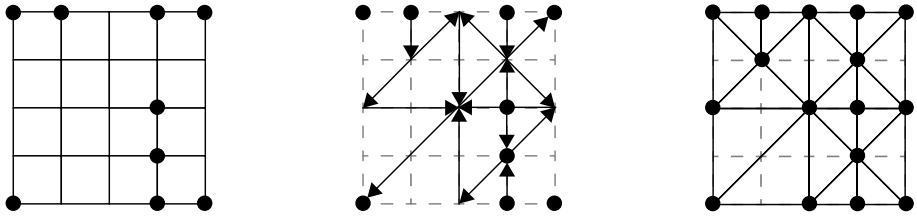


Figure 9.3 Restricted quadtree triangulation using the dependency graph

The author proposed two algorithms that resolve all the dependencies and construct a RQT in linear time:

1. Using a top-down approach, the first algorithm starts from the root node and visits each node in depth-first order. If the error associated with a vertex *i* is larger than a user-defined threshold, the vertex *i* is selected. In order to satisfy the restriction criterion the algorithm must resolve all the dependencies associated with vertex *i*. Since the dependencies must be resolved it is possible that a vertex is visited more than once, but the author has proved that the algorithm is still linear.
2. Using a bottom-up approach, the second algorithm starts from the leaf nodes. If the error associated with a vertex *i* is high enough, the vertex *i* is selected. The algorithm then marks the vertices in the dependent list associated with *i*, collects them. If the algorithm visit the vertices in the correct order from the vertices with the most dependencies to the vertex with the least dependencies (the central vertex), then every vertex is checked exactly once.

9.2.2 Error Computation

In the algorithms described in the previous section a vertex is selected for a triangulation if its associated error was larger than a user-defined threshold. It is therefore necessary to compute an error value per vertex. The author defined the error norm for a vertex *i* at level *l* as follows:

$$e_{T_{l-1}}(i) = \max_{j \in \zeta(i),\, t \in \mathrm{Cov}(i)} d(i, t) \qquad \text{(EQ 9.22)}$$

The set $\zeta(i)$ contains all the vertices in the quadtree that depends on the vertex i

$$\zeta(i) = \left\{ j \in \bigcup_{k=0}^{\log(n)-1} L_l \,\middle|\, \exists v_1, \ldots, v_m; i \leftarrow v_1 \leftarrow \ldots \leftarrow v_m \leftarrow j \right\} \cup \{ i \} \qquad \text{(EQ 9.23)}$$

The term T_{l-1} describes the best triangulation that does not include i, which is defined as

$$T_{l-1} = \left\{ \mathrm{RQT}\left(\bigcup_{k=0}^{l-1} L_k \right) \right\} \quad \text{(EQ 9.24)}$$

if the vertex *i* is the center vertex in a node at level *l*, otherwise as

$$T_{l-1} = \left\{ \mathrm{RQT}\left(\bigcup_{k=0}^{l-1} L_k \cup L_l^c \right) \right\} \quad \text{(EQ 9.25)}$$

The set $\mathrm{Cov}(i)$ contains all the triangles that are affected by the selection of the vertex *i* in the triangulation. More precisely it is defined as

$$\mathrm{Cov}(i) = \{ t \in T_{l-1} \,|\, P(t) \cap P(i) \neq \varnothing \} \quad \text{(EQ 9.26)}$$

The function $P(x)$ computes the parameter domain of the vertex i and of the triangle t.

Finally the function $d(i,t)$ measures the distance between the vertex i and the triangle t. Since the data is considered to be an height field, the vertical distance is measured.

In simpler words, Eq. 9.22 measures the error introduced by the removal of the vertex i. In order to measure the correct error, not only the vertical distance between i and the triangulation is computed, but also the vertical distance between all the points that depends on i and the triangulation. This is necessary, since if i is not present, then all the points that depends on i cannot be present. The drawback of this implementation is that the complexity of the algorithm is larger than $O(n \cdot \log n)$.

9.2.3 Fast Triangulation

One interesting enhancement of the representation presented by Pajarola is that the author is able to construct triangle strips very easily. Once vertices are selected and a restricted triangulation is built, it is possible to circle counterclockwise through the mesh and visit all the triangles exactly once. The two masks used to describe the path through the mesh are shown in Figure 9.4.

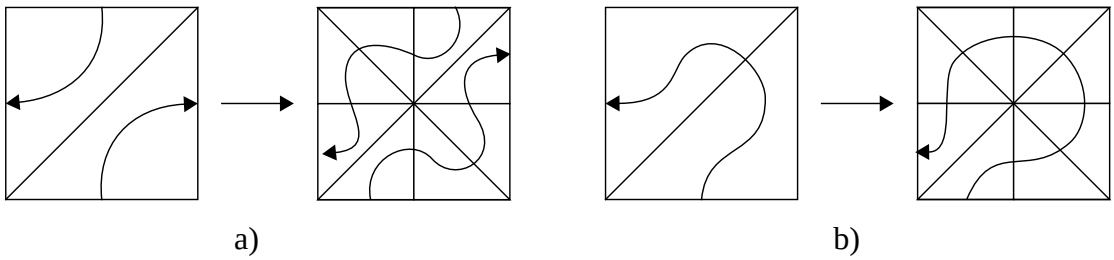


Figure 9.4 Masks a) and b) allow to construct a path through the RQT

Figure 9.5 illustrates how the use of these two masks allows to compute a strip for the example given in Figure 9.3.

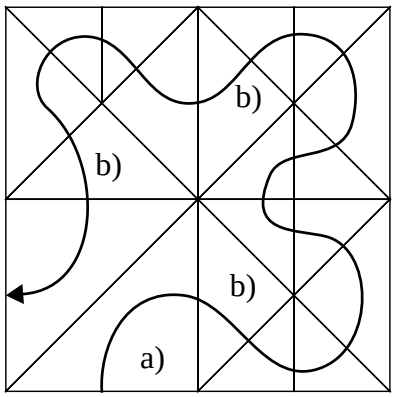


Figure 9.5 Generation of the triangle strip
a) Mask a has been used
b) Mask b has been used

The algorithm is very fast: the strip can be computed in linear time. The triangulation algorithm discussed in this section can only be applied on grids of size $(2^n + 1) \times (2^n + 1)$. If this is not the case, then it is necessary to build other masks to handle special cases that can occur at the boundary.

9.2.4 Other Properties of the Representation

The author described other properties of the RQT representation he developed. By ordering the vertices cleverly it is possible, for example, to construct a *progressive mesh* sequence with continuous level of details

$$L_0, L_1^c, L_1 \backslash L_1^c, \ldots, L_{\log(n-1)}^c, L_{\log(n-1)} \backslash L_{\log(n-1)}^c \qquad \text{(EQ 9.27)}$$

If the first l vertices from this sequence are used, then the result is a RQT, and furthermore this is the best representation with l vertices in the sense of the error norm presented in Section 9.2.2.

Since the quadtree segments the data spatially, it is possible to construct *view dependent continuous level of details*, simply by assigning different error thresholds to different patches of the surface, usually quadtree nodes. Since the algorithm still generates a RQT, there are no discontinuities between patches with different thresholds.

9.3 Evaluation

In this section the DSI algorithm and the RQT implementation of Pajarola will be evaluated using the criteria specified in Section 2.

- *Modeling of two-manifolds with boundaries*: The DSI algorithm can be applied to two-manifold surfaces, the algorithm does not make assumptions on the type or connectivity of the meshes it is handling.

  The RQT representation can only handle grids, and therefore it cannot model arbitrary two-manifold surfaces. However it is possible to extend the basic RQT algorithm to handle grids with irregular boundaries and holes.

- *Computation of surface-surface intersections*: Both representations do not have the information required to compute the intersections between surfaces. A hierarchical bounding box data structure could be used to accelerate the computations. Since the RQT is implemented as a quadtree, it would be possible to include the bounding boxes directly in the representation.

- *Scalable representation*: The RQT representation is scalable, since it constructs a multiresolution hierarchy on top of the full resolution surface. The representation satisfies these properties:

  - The representation is compact, since the overhead introduced by the quadtree is in the worst case linear.
  - The construction of the quadtree proposed by Pajarola is not very efficient, since the computation of the error has a complexity larger than $O(n \cdot \log n)$. However it would be possible to use simpler error models in order to reduce the time needed to construct the quadtree.
  - An error bounded approximation of the full resolution surface can be constructed in linear time, since the algorithms presented in Section 9.2.1 to resolve the dependencies and the stripping algorithm described in Section 9.2.3 have a linear complexity.

  The DSI algorithm is not used to construct a multiresolution representation of a surface, and therefore it is not scalable. The DSI is not to be interpreted as a stand alone representation, but as a tool to use in conjunction with other representations. The algorithm is fast and can deform a mesh to satisfy a set of constraints in $O(n \cdot k)$, where n is the number of vertices in the mesh, and k is the number of iterations needed to converge to the solution.

- *Modeling of non-manifold singularities, tears and cracks*: The RQT representation cannot model general non-manifold surfaces, since its connectivity is constrained to regular grids.

  The DSI could be used to enforce linear constraints on a non-manifold model, since the algorithm does not make any assumption on the mesh it is working on, and since no parametrization is needed.

  Curves could be embedded in the full resolution meshes stored in the two representations using a separate data structure.

- *Error modeling*: The error norm Pajarola used for his representation is very similar to the norms used by the mesh-based methods discussed in Section 7. The representation can construct a representation that satisfies an error threshold specified by the user, but more vertices than necessary would be used.

The DSI algorithm does not construct approximations of the mesh. The algorithm will construct meshes that satisfy a set of user-defined constraints. If there are more constraints than degrees of freedom, then the DSI is going to construct a solution that minimizes the two-norm of the residual.

- *Smoothness of the surface*: The RQT works on the piecewise linear representation of surfaces, and therefore it is not capable to represent or handle smooth surfaces.

  The DSI works on polygonal meshes (usually a triangle mesh), but it is possible to impose higher order constraints, thus generating "smooth" piecewise linear surfaces. This is similar to the signal processing papers, which smooth piecewise linear meshes.

- *Multiresolution editing*: The RQT constructs a multiresolution representation on top of surfaces, but since the author did not use local frames and smoothing operators to store the vertices, it is not possible to edit the surface at different levels of resolution effectively.

  The DSI algorithm does not build a multiresolution representation, so multiresolution editing is not supported.

- *Surface fitting*: The DSI algorithm is often used in geoscience to fit surfaces, for which the geometric information is not fully known. However the algorithm does not construct the connectivity of the surface: this information must be provided to the algorithm.

  The RQT representation does not fit grids through a cloud of points. It would be possible however to run a fitting algorithm that generates a grid as a pre-computation, feeds the grid to RQT, which will generate the multiresolution representation.

- *Support of local high variation in the curvature of the surface*: Both representations work on piecewise linear surfaces, so they can handle local high variation in the curvature.

- *Changes of the surface over time*: Since the RQT representation works with regular grids, the microtopology of the grid is not allowed to change over time. Consequently the representation can handle changes over time well by specifying a function $f(t)$ at each vertex position that returns the geometric position $f(t_i) = (x_i, y_i, z_i)$ for time t_i. Changes in the resolution of the grids can be modeled easily as well.

  The DSI is not a representation, so if the surface changes over time, the DSI algorithm will probably be applied at all times t_i independently. If the functions defined at the vertices at different time steps are not independent, then the extension to the DSI discussed in the original paper is capable to handle the dependencies if and only if the topology remains the same.

10 Conclusions

In Section 4 through Section 8 we analyzed many different representations, and we evaluated them using the requirements described in Section 2. From the evaluation it is clear that none of the representation satisfies all the requirements.

Comparing the different representations and their properties we conclude that:

- *Wavelets* represent a very powerful tool for surface representation. The most important properties of wavelets are summarized below:

+ modeling of the error	- Topology limited to regular grids
+ scalable representation	- Smoothness set by the basis function
+ surface intersection	- Global parametrization required
+ compact representation	
+ multiresolution edit	
+ changes of the surface over time	

  Due to the rigorous formulation of the wavelet framework many requirements can be met with simple and elegant solutions. However, the rigorousness of this framework limits the topology of the surfaces to simple height fields represented over regular grids, and it requires a global parametrization of the surface.

- The *H-Spline* is a very simple and elegant framework that constructs a multiresolution representation of B-Spline surfaces. Advantages and disadvantages of H-Spline are summarized in the following list:

+ scalable representation	- Smoothness set by the basis function
+ multiresolution edit	- Cannot model non-manifold
+ surface intersection	- Global parametrization required
+ some control over the error	
+ scalable representation	
+ changes of the surface over time	

  Since the H-Spline representation is a predecessor of the B-Spline wavelets it shares most of the advantages of the wavelets, but it is not always easy to make use of these advantages. For example it is possible to construct tools that measure error for some error norms, but it is not as rigorous as for the wavelets. The H-Spline representation is scalable, but some control points are usually redundant.

  H-Splines allow to construct surfaces that are more complex than regular grids, and in this respect they are more general than wavelets.

- *Subdivision schemes* construct smooth surfaces in the limit from piecewise linear meshes with arbitrary connectivity. The advantages and disadvantages of this representations are listed below:

+ modeling of two-manifolds	- subdivision connectivity
+ some control over the error	- surface intersection
+ scalable representation (SWT)	- changes of the surface over time
+ multiresolution edit (SWT)	

+ modeling of smooth and non-smooth surfaces
+ only a local parametrization is required

Subdivision schemes can model two-manifold surfaces with boundary, but they are limited in the connectivity of the mesh, since they generate meshes with subdivision connectivity. Subdivision schemes can model non-smooth regions of the surface well using special operators.

Surface-surface intersections can be computed only using bounding box algorithms, special algebraic methods for subdivision surfaces need to be constructed.

- *Mesh based methods* identify surfaces with the piecewise linear meshes. Advantages and disadvantages of this approach are investigated in the following list:

+ modeling of two-manifolds	- error modeling
+ scalable representation	- don't model smooth surfaces
+ modeling of non-smooth surfaces	- multiresolution edit
+ no parametrization needed	- changes of the surface over time
	- surface intersection

Mesh based methods are very flexible and can build scalable multiresolution representations of two-manifolds and usually non-manifold surfaces. They work with piecewise linear surfaces, so non-smooth regions can be modeled well.

The drawback of these algorithms is that they lack most of the mathematical background of the other approaches. As a result error modeling is much less rigorous, and heuristics are used to model the error; since the representations work with piecewise linear functions smooth surfaces cannot be modeled, and multiresolution editing tools do not generate intuitive edits.

- Representations based on *signal processing tools* construct special operators that minimize the curvature of a mesh. The main features of these representations are described in the following list:

+ modeling of two-manifolds	- surface intersection
+ scalable representation	- error modeling
+ multiresolution edit	- changes of the surface over time
+ modeling of smooth and non-smooth surfaces	
+ only a local parametrization is required	

The fairing operator allows to smooth meshes with arbitrary connectivity. As a result, smooth surfaces can be modeled, and multiresolution editing tools perform well. Note that no C^n continuity can be guaranteed by these representations.

From the summary of the representations outlined above the following conclusions can be drawn:

1. The strong mathematical background of the wavelet representations allow to model error and to construct compact multiresolution representations for surfaces. Furthermore, since surfaces are represented as functions in R^2 it is possible to represent smooth surfaces and to build multiresolution editing tools.

2. Mesh based methods are more flexible: every representation presented in this report can model at least two-manifolds, and most of them can model non-manifolds as well. These representations are usually also scalable, but they store redundant information.
3. The signal processing representations lie in between the rigorous mathematical approach of wavelets and the flexible approach of mesh based methods. These representations work on meshes with arbitrary microtopology, and use discrete operators generated from well-understood signal processing theory to fair surfaces.

Based on this conclusions we describe some possible directions for future works:

1. Methods based on signal processing are very interesting, since they unify some of the properties of wavelet-like approaches and mesh-based approaches.
   It would be interesting to continue in a similar direction and try to build wavelet-like operators, for which part of the classic wavelet theory could be extended easily. As an example it would be very interesting to construct operators that work on meshes with arbitrary connectivity for which a solid error analysis could be built.
2. Another interesting extension of the current representations is the ability to model non-manifold surfaces. A particularly interesting extension in this direction could be the modeling of multiresolution representations of intersection curves, as well as any other curve embedded in a surface.
3. One other possible direction for research is to extend some of the ideas presented in this paper, particularly the signal processing operators, to three dimensions.

Acknowledgments

We would like to thank Richard hammersley, Karen Lu and Christoph Ramshorn for the help they provided in reviewing the report and suggesting changes and improvements. We would like to thank *Schlumberger Austin Product Center* for the financial support.

11 References

[1] F. L. André Guéziec, Gabriel Taubin and W. Horn. "Converting sets of polygons to manifold surfaces by cutting and stitching." In *Proceedings of the IEEE Visualization 1998*. IEEE Visualization, Oct. 1998.

[2] G. Battle. "A block spin construction of ondelettes, Part I: Lemari'e functions." *Communications on Mathematical Physics*, 110:601, 1987.

[3] J. Benedetto and D. Walnut. "Gabor frames for L^2 and related spaces." In J. Benedetto and M. Frazier, editors, *Wavelets: Mathematics and Applications*. pub-crc, pub-crc:adr, 1993.

[4] J. J. Benedetto and M. Frazier, editors. *Wavelets: Mathematics and Applications*. Studies in Advanced Mathematics. CRC Press, 2000 Corporate Blvd., Boca Raton, FL 33431, USA, 1992.

[5] G. Bonneau. "Multiresolution analysis with non-nested spaces." In *Dagstuhl seminar on Geometric Modeling*, 1996.

[6] G. Bonneau. "Hierarchical decomposition of datasets on irregular surface meshes." In *Proceedings of CGI'98, Hannover*, 1998.

[7] G. Bonneau, S. Hahmann, and G. M. Nielson. "BLaC-Wavelets: A multiresolution analysis with non-nested spaces." In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.

[8] G.-P. Bonneau. "Multiresolution analysis on irregular surface meshes." *IEEE Transactions on Visualization and Computer Graphics*, 4(4), Oct. 1998.

[9] E. Catmull and J. Clark. "Recursively generated B-spline surfaces on arbitrary topological meshes." *Computer-Aided Design*, 10:350–355, Sept. 1978.

[10] C. K. Chui. *An Introduction to Wavelets*, volume 1 of *Wavelet Analysis and its Applications*. Academic Press, Inc., 1992.

[11] C. K. Chui and J. Z. Wang. "An analysis of cardinal spline-wavelets." Technical report, CAT Report 231, Texas A&M University, 1990. thk.

[12] A. Cohen. *Ondelettes, analysis multirésolutions et traitement numérique du signal*. PhD thesis, Université Paris IX Dauphine, 1990.

[13] A. Cohen and I. Daubechies. "Orthonormal bases of compactly supported wavelets III. better frequency resolution." *SIAM Journal on Mathematical Analysis*, 24(2):520–527, Mar. 1993.

[14] A. Cohen, I. Daubechies, and J. Feauveau. "Bi-orthogonal bases of compactly supported wavelets." *Comm. Pure and Appl. Math.*, 45:485–560, 1992.

[15] I. Daubechies. "Orthonormal bases of compactly supported wavelets." *Comm. Pure Applied Math.*, XLI(41):909–996, Nov. 1988.

[16] I. Daubechies. *Ten Lectures on Wavelets*, volume 61 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[17] I. Daubechies. "Orthonormal bases of compactly supported wavelets: II. variations on a theme." *SIAM J. Math. Anal.*, 24:499–519, 1993.

[18] I. Daubechies, I. Guskov, P. Schröder, and W. Sweldens. "Wavelets on irregular point sets." *Phil. Trans. R. Soc. Lond. A*, To be published.

[19] I. Daubechies and W. Sweldens. "Factoring wavelet transforms into lifting steps." Technical report, Bell Laboratories, Lucent Technologies, 1996.

[20] J. Daugman and C. Downing. "Gabor wavelets for statistical pattern recognition." In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 414–419. MIT Press, Cambridge, Massachusetts, 1995.

[21] C. de Boor, K. Hollig, S. Riemenschneider, and A. Ron. "Box splines." *SIAM Review*, 37(2):275–??, June 1995.

[22] T. DeRose, M. Kass, and T. Truong. "Subdivision surfaces in character animation." In M. F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings- 1998, pages 85–94, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison Wesley.

[23] T. D. DeRose, M. Lounsbery, and J. Warren. "Multiresolution analysis for surfaces of arbitrary topological type." Technical Report 93-10-05, Department of Computer Science and Engineering, University of Washington, 1993.

[24] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. "Implicit fairing of irregular meshes using diffusion and curvature flow." In *SIGGRAPH '99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[25] D. Doo. "A subdivision algorithm for smoothing down irregularly shaped polyhedrons." In *Proced. Int'l Conf. Ineractive Techniques in Computer Aided Design*, pages 157–165, 1978. Bologna, Italy, IEEE Computer Soc.

[26] D. Doo and M. Sabin. "Behaviour of recursive division surfaces near extraordinary points." *Computer-Aided Design*, 10:356–360, Sept. 1978.

[27] A. Dreger, M. H. Gross, and J. Schlegel. "Multiresolution triangular B-spline surfaces." Technical Report 279, Computer Science Department, ETH Zürich, Dec. 1997.

[28] N. Dyn, S. Hed, and D. Levin. "Subdivision schemes for surface interpolation." *Workshop in Computational Geometry*, pages 97–118, 1993.

[29] N. Dyn, D. Levin, and J. A. Gregory. "A butterfly subdivision scheme for surface interpolation with tension control." *ACM Transactions on Graphics*, 9(2):160–169, Apr. 1990.

[30] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. "Multiresolution analysis of arbitrary meshes." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[31] M. Eck, T. D. DeRose, T. DuChamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. "Multiresolution analysis of arbitrary meshes." Technical Report TR-95-01-02, University of Washington, Department of Computer Science and Engineering, Jan. 1995.

[32] M. Eck and H. Hoppe. "Automatic reconstruction of B-Spline surfaces of arbitrary topological type." In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 325–334. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[33] G. Fernández, S. Periaswamy, and W. Sweldens. "LIFTPACK: A software package for wavelet transforms using lifting." In M. A. Unser, A. Aldroubi, and A. F. Laine, editors, *Wavelet applications in signal and image processing IV*, volume 2825 of *Proceedings of SPIE*, 1996.

[34] D. Forsey and R. H. Bartels. "Surface fitting with hierarchical splines." *ACM Transactions on Graphics*, 14(2):134–161, Apr. 1995.

[35] D. Forsey and D. Wong. “Multiresolution surface reconstruction for hierarchical B-splines.” In W. Davis, K. Booth, and A. Fourier, editors, *Proceedings of the 24th Conference on Graphics Interface (GI-98)*, pages 57–64, San Francisco, June18–20 1998. Morgan Kaufmann Publishers.

[36] D. R. Forsey and R. H. Bartels. “Hierarchical B-spline refinement.” volume 22, pages 205–212, Aug. 1988.

[37] M. Garland and P. S. Heckbert. “Surface simplification using quadric error metrics.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[38] M. H. Gross and R. Koch. “Visualization of multidimensional shape and texture features in laser range data using complex-valued gabor wavelets.” *IEEE Transactions on Visualization and Computer Graphics*, 1(1):44–59, Mar. 1995. Figures: ftp://ftp.inf.ethz.ch/pub/publications/papers/is/cg/ieee-tvcg95/ieee-tvcg95.% fig_xx.gif.

[39] M. H. Gross, O. G. Staadt, and R. Gatti. “Efficient Triangular Surface Approximations Using Wavelets and Quadtree Data Structures.” *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.

[40] I. Guskov. “Multivariate subdivision schemes and divided differences.” Technical report, Program for Applied and Computational Mathematics, Princeton University NJ 08544, 1998.

[41] S. Gysel. “Hierarchical representations and manipulation of geological layers.” Unpublished report, Department of Computer Science, ETH Zurich, Switzerland, 1998.

[42] H. Hoppe. “Surface reconstruction from unorganized points.” Technical Report TR-94-06-01, University of Washington, Department of Computer Science and Engineering, June 1994.

[43] H. Hoppe. “Progressive meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[44] H. Hoppe. “View-dependent refinement of progressive meshes.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[45] H. Hoppe. “Efficient implementation of progressive meshes.” *Computers & Graphics*, 22(1):27–36, Feb. 1998. ISSN 0097-8493.

[46] P. S. Igor Guskov, Wim Sweldens. “Multiresolution signal processing for meshes.” In *SIGGRAPH ’99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[47] J. L. Jiankun Li and C.-C. J. Kuo. “Progressive compression of 3d graphic models.” In *IEEE International Conference on Multimedia Computing and Systems*. IEEE, June 1997.

[48] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. “Interactive multi-resolution modeling on arbitrary meshes.” In M. F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings- 1998, pages 105–114, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison Wesley.

[49] V. Krishnamurthy and M. Levoy. “Fitting smooth surfaces to dense polygon meshes.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 313–324. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[50] S. Krishnan and D. Manocha. “An efficient surface intersection algorithm based on lower dimensional formulation.” Technical Report TR94-062, Department of Computer Science, University of North Carolina - Chapel Hill, Nov. 22 1994.

[51] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. “MAPS: Multiresolution adaptive parameterization of surfaces.” In M. F. Cohen, editor, *Computer graphics: proceedings: SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, Computer Graphics -proceedings- 1998, pages 95–104, New York, NY 10036, USA and Reading, MA, USA, 1998. ACM Press and Addison Wesley.

[52] P. G. Lemari’e. “Ondelettes ‘a localisation exponentielle.” *J. Math. Pures Appl. (9)*, 67:227–236, 1988. n.

[53] A. Levin. “Analysis of combined subdivision schemes for the design of surfaces satisfying boundary conditions.” Unpublished report, 1998.

[54] A. Levin. “Combined subdivision schemes for the design of surfaces satisfying boundary conditions.” 1999.

[55] C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, Utah University, 1987.

[56] M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1994.

[57] M. Lounsbery, T. D. DeRose, and J. Warren. “Multiresolution analysis for surfaces of arbitrary topological type.” *ACM Transactions on Graphics*, 16(1):34–73, Jan. 1997.

[58] K. Lu, E. Schoen, J. Salim, Y. Cudennec, and C. Ramshorn. “Common model builder – a toolkit for multidisciplinary geoscience modeling applications.” GOCAD ENSG Conference 3D Modeling of Natural Objects: A Challenge for the 2000’s, June 1998.

[59] J.-L. Mallat. “Discrete smooth interpolation in geometric modeling.” *CAD Computer Aided Design*, 24(4):178–192, 1992.

[60] S. Mallat. *Multiresolution Representations and Wavelets*. PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, 1988. Also Grasp Lab Tech report 153.

[61] S. G. Mallat. “A theory for multiresolution signal decomposition: The wavelet representation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[62] D. Manocha and S. Krishnan. “Algebraic pruning: a fast technique for curve and surface intersection.” *Computer Aided Geometric Design*, 14(9):823–845, 1997. ISSN 0167-8396.

[63] W. S. Massey. *A Basic Course in Algebraic Topology*. Springer Verlag, 1997.

[64] Y. Meyer. *Wavelets and Operators*. Cambridge University Press, 1992.

[65] Y. Meyer. *Wavelets: Algorithms and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1993.

[66] R. Pajarola. “Large scale terrain visualization using the restricted quadtree triangulation.” In *Proc. Visualization 1998*. IEEE, 1998.

[67] J. Popovic and H. Hoppe. “Progressive simplicial complexes.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[68] E. Quak and N. Weyrich. “Decomposition and reconstruction algorithms for spline wavelets on a bounded interval.” *BIT Numerical Mathematics*, 1:217–231, 1994.

[69] E. Quak and N. Weyrich. “Algorithms for spline wavelet packets on an interval.” *BIT Numerical Mathematics*, 37(1):76–95, Mar. 1997.

[70] S. Assa, R. Hammersley, K. Lu. “Geometric modeling with a multiresolution representation.” Aug. 1999. held in Vancouver, Canada, 06-11 August 1995.
[71] F. Riesz and B. Sz.-Nagy. *Functional Analysis*. Ungar, New York, 1955. L. F. Boron, Translator.
[72] C. Ramshorn, S. Assa, G. Celniker. “Feature-based geometric modeling for geoscience.” GOCAD ENSG Conference 3D Modeling of Natural Objects: A Challenge for the 2000’s, June 1998.
[73] P. Schröder and W. Sweldens. “Spherical wavelets: Efficiently representing functions on the sphere.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 161–172. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.
[74] W. J. Schröder, J. A. Zarge, and W. E. Lorensen. “Decimation of triangle meshes.” volume 26, pages 65–70, July 1992.
[75] T. W. Sederberg, J. Zheng, D. Sewell, and M. Sabin. “Non-uniform recursive subdivision surfaces.” In M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 387–394. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.
[76] G. E. Shilov. *Elementary Functional Analysis*. Dover Publications, Mineola, 1974. R. A. Silverman, Translator.
[77] O. G. Staadt and M. H. Gross. “Progressive tetrahedralizations.” In *Proceedings IEEE Visualization ’98*, pages 397–402. IEEE, 1998.
[78] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. “Wavelets for computer graphics: A primer, part 1.” *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995.
[79] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. “Wavelets for computer graphics: Part 2.” *IEEE Computer Graphics and Applications*, 15(4):75–85, July 1995.
[80] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, MA, 1996.
[81] W. Sweldens. “The lifting scheme: A construction of second generation wavelets.” *SIAM Journal on Mathematical Analysis*, 29(2):511–546, Mar. 1998.
[82] W. Sweldens. “The lifting scheme: A custom-design construction of biorthogonal wavelets.” 1998.
[83] W. Sweldens. “The lifting scheme: A new philosophy in biorthogonal wavelet construction.” 1998.
[84] W. Sweldens. “Wavelets and the lifting scheme: A 5 minute tour.” 1998.
[85] W. Sweldens and P. Schröder. “Building your own wavelets at home.” In “Wavelets in Computer Graphics”, ACM SIGGRAPH Course Notes, 1996.
[86] G. Taubin. “A signal processing approach to fair surface design.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.
[87] R. S. W. Dahmen. “Wavelets over manifolds i: Construction and domain decomposition.” *IGMP Report*, 149, 1998.
[88] D. Zorin, H. Biermann, and A. Levin. “Piecewise smooth subdivision surfaces with normal control.” Technical Report TR1999-781, New York University, Feb. 26, 1999.
[89] D. Zorin, P. Schröder, and W. Sweldens. “Interpolating subdivision for meshes with arbitrary topology.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 189–192. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.
[90] D. Zorin, P. Schröder, and W. Sweldens. “Interactive multiresolution mesh editing.” In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 259–268. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.